



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO

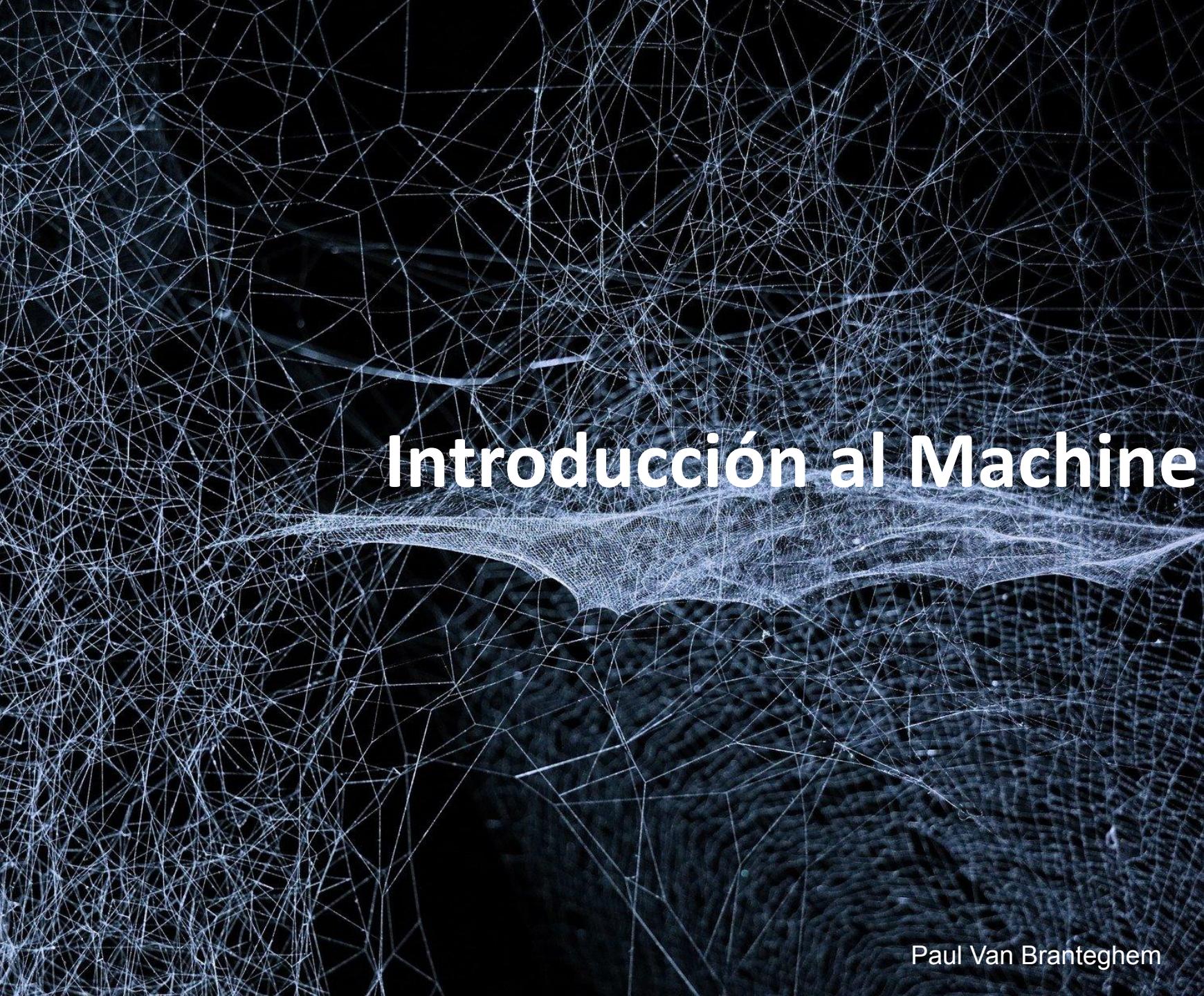


Escuela de
organización
industrial



Redes neuronales 2023

Paul Van Branteghem



Introducción al Machine Learning

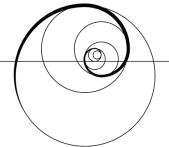
Paul Van Branteghem

Sobre mí

PAUL VAN BRANTEGHEM

- CTO & VP & co-funder de Big Onion y co-founder de Spain AI
- Físico con máster en IA avanzada, meteorología, Programa Superior en Analítica Digital y CRM
- Profesor en diferentes programas de Data Science / Big Data
- Experiencia en Siemens-Gamesa, en el departamento de IA de BBVA, en Deloitte Digital y en StratioBD

BIG ONION



Co-founder

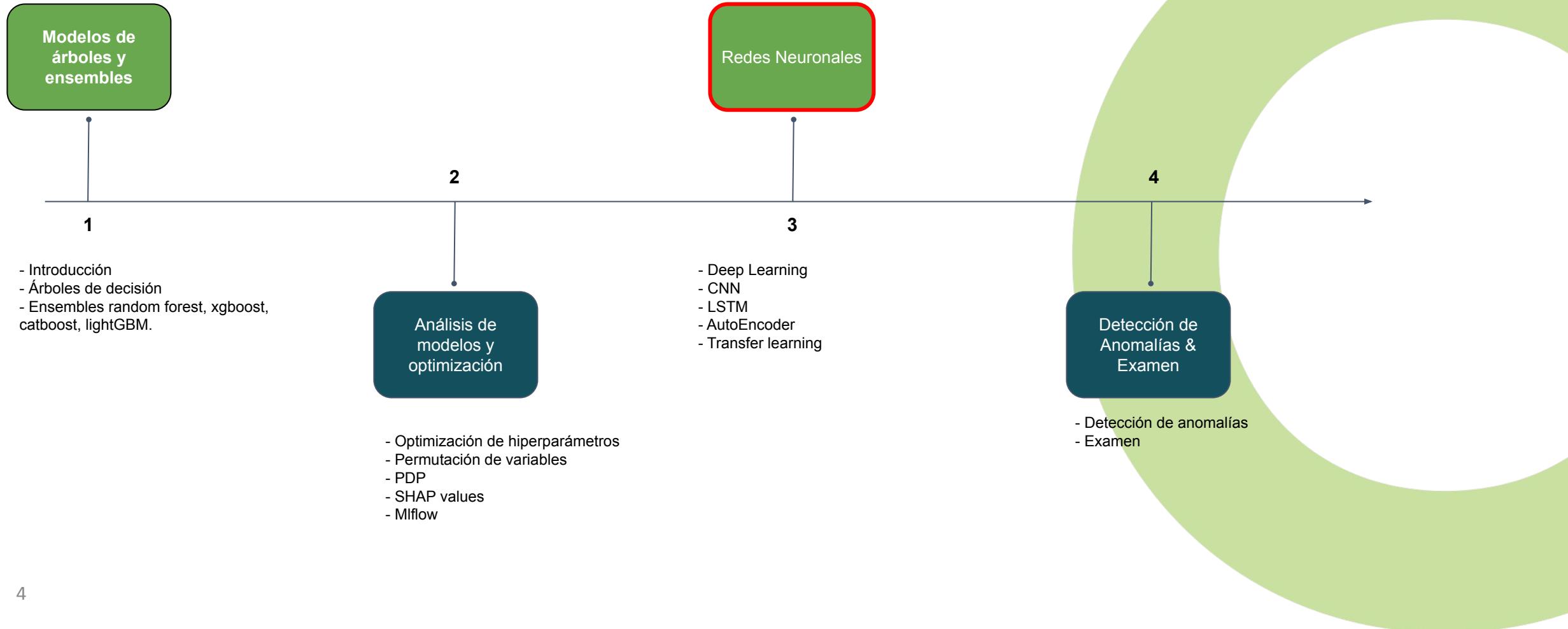


/in/paulvanlorenzo



paul@spain-ai.com

Temario del programa

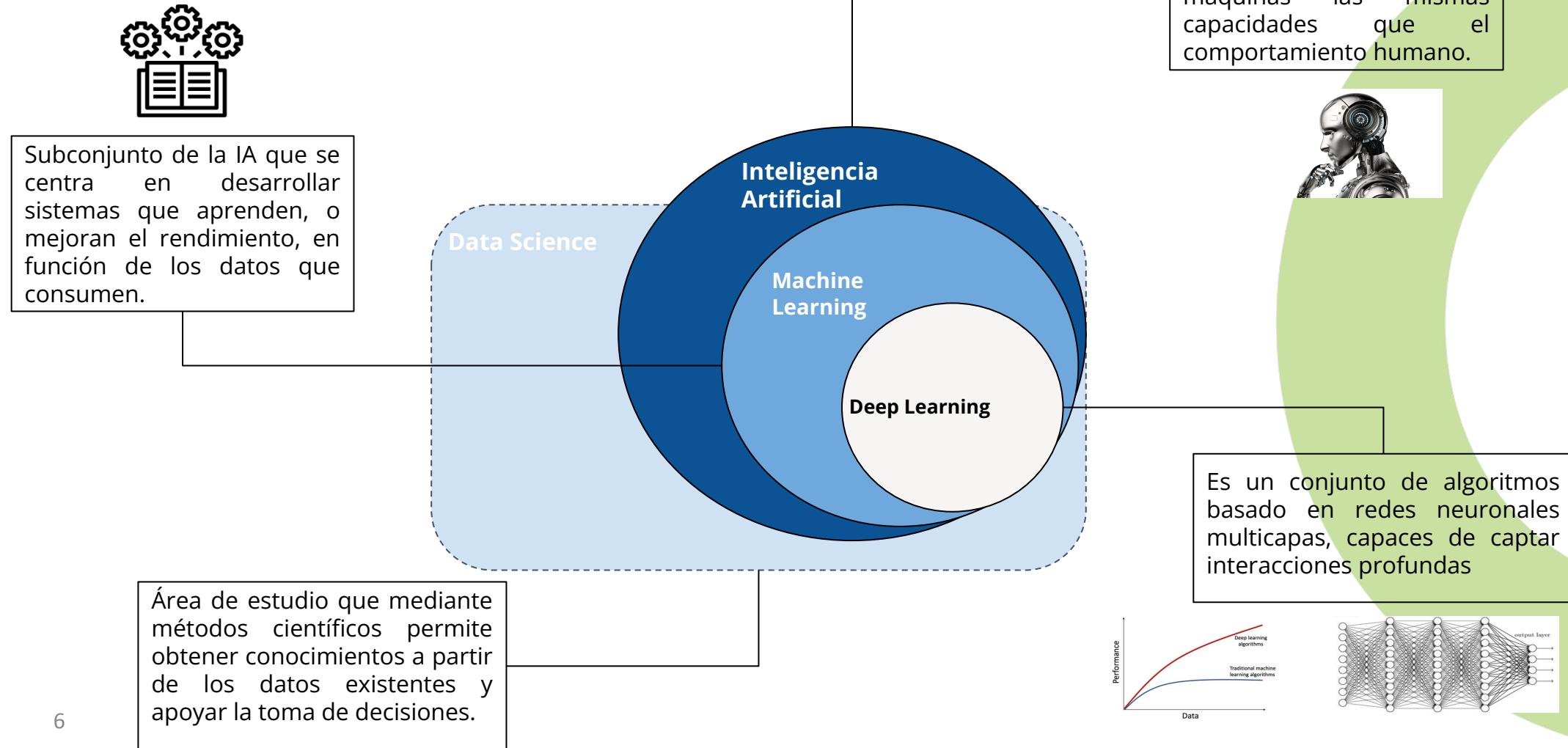


Índice de la sesión

- **Recap**
 - **Introducción**
 - Árboles de decisión y ensembles
 - Hyperparameter tuning
 - Explicabilidad
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- Modelos de Deep Learning

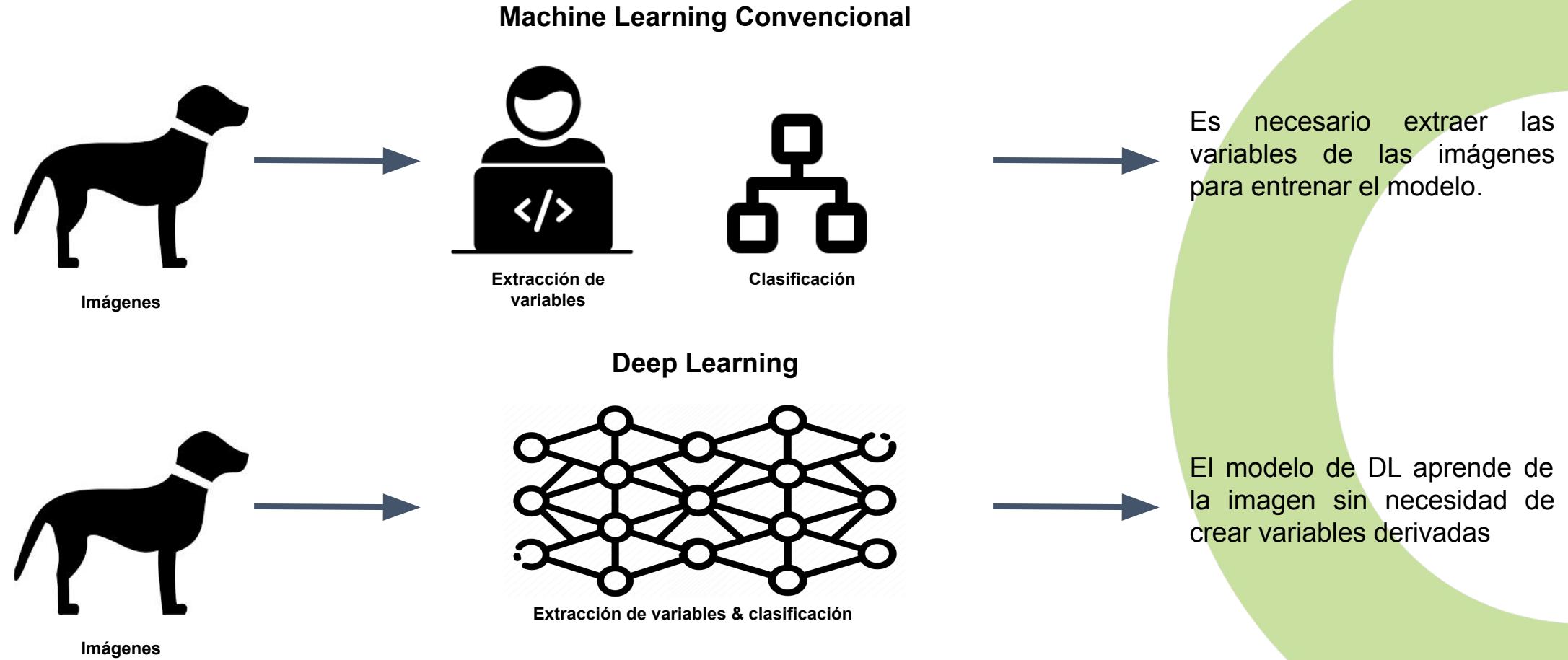
Introducción a la IA

Es por ello saber diferenciar correctamente Inteligencia Artificial, Machine Learning y Deep Learning.



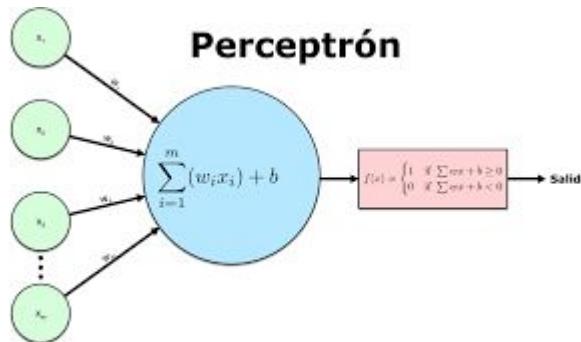
Modelos de Deep Learning

¿ Cuál es la diferencia respecto al machine learning convencional?

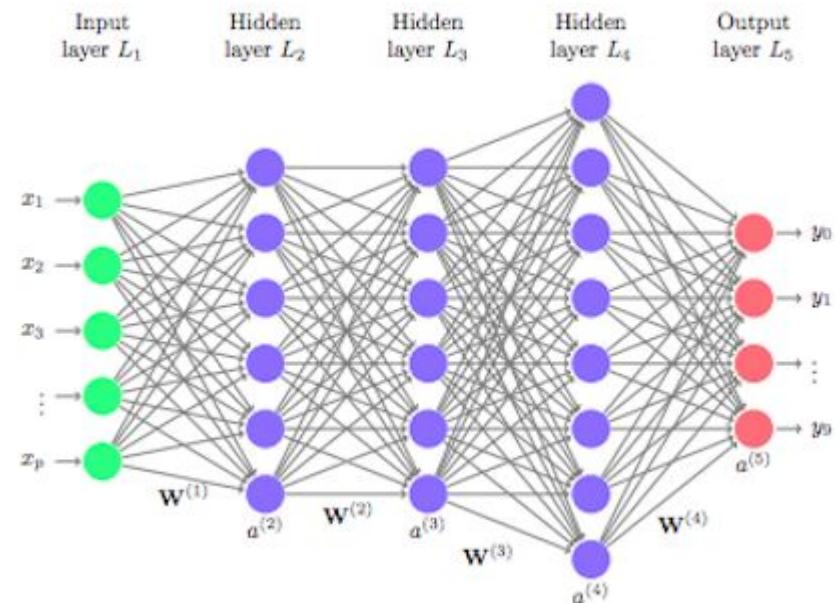
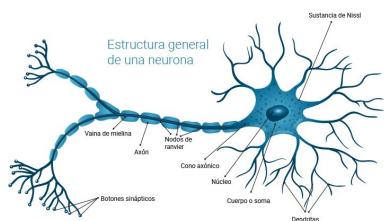


Modelos de Deep Learning

El deep learning permite obtener interacciones profundas entre los datos mediante redes neuronales.

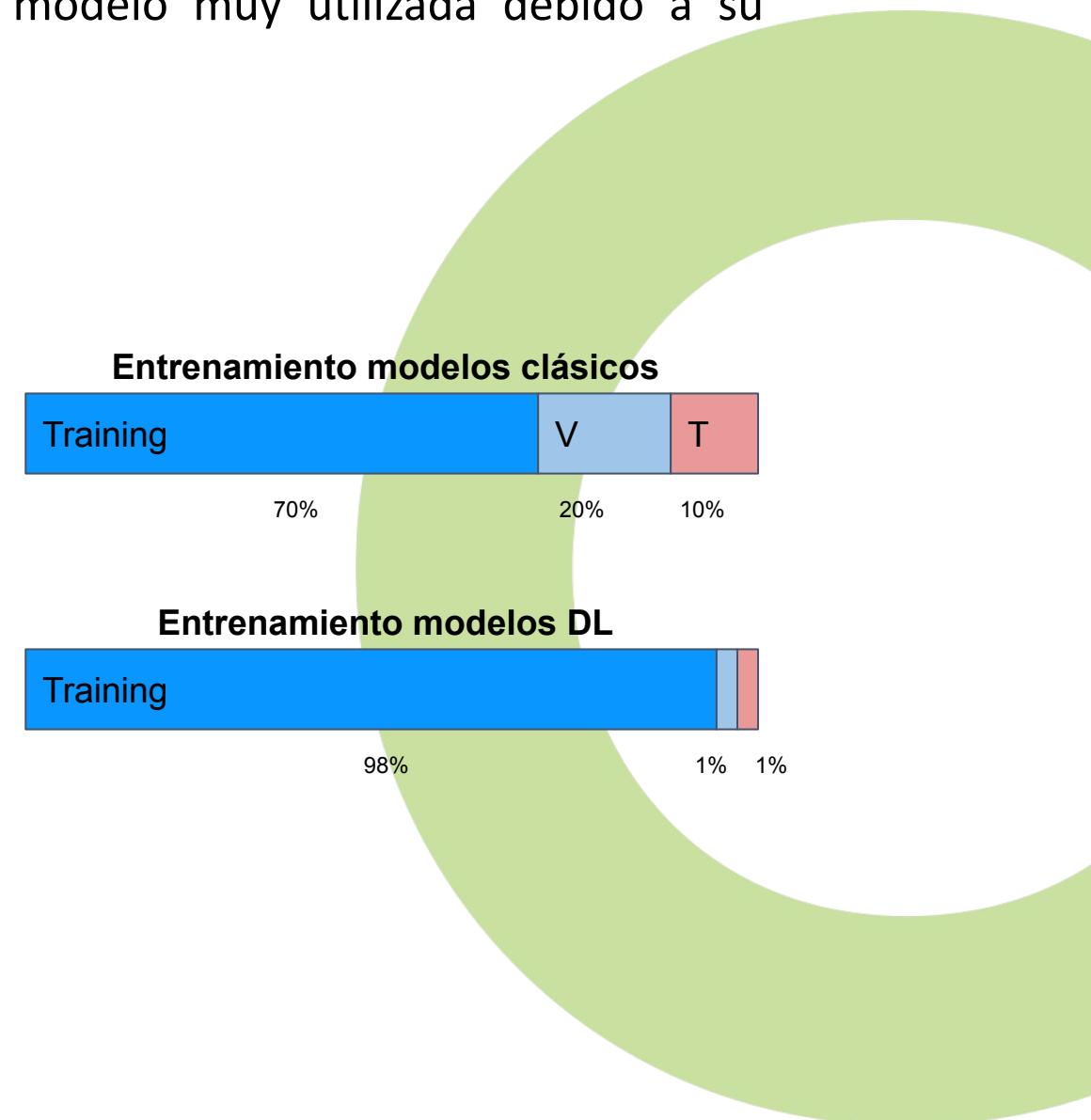
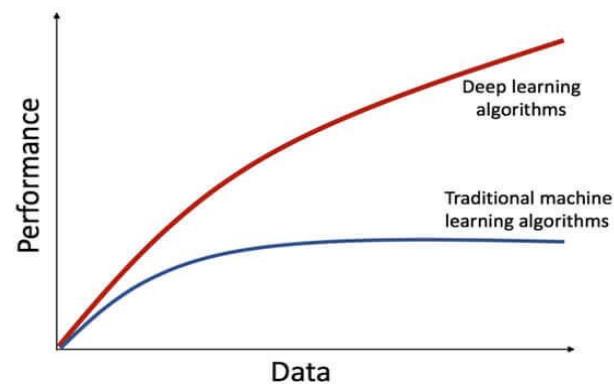
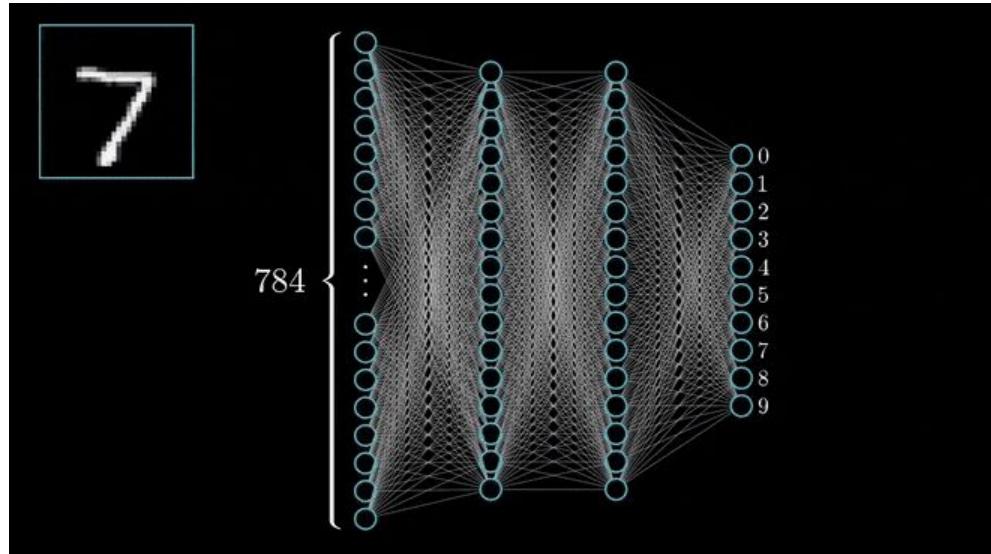


La unidad fundamental de las redes neuronales es el perceptrón.



Modelos de Deep Learning

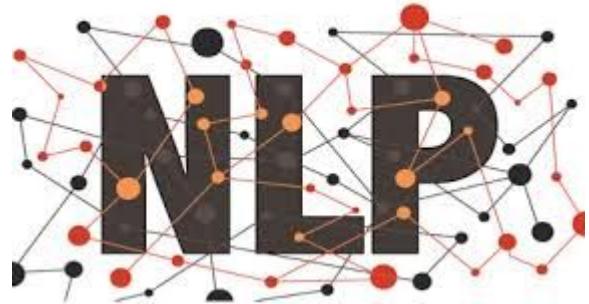
El Deep Learning se ha convertido en una tipología de modelo muy utilizada debido a su performance con grandes cantidades de datos.



Modelos de Deep Learning

El performance es mucho mejor con los algoritmos de Deep Learning cuando trabajamos con imágenes (CV) y también con procesamiento del lenguaje natural (NLP).

NATURAL LANGUAGE PROCESSING



En casos como **NLP** y **Computer Vision** los modelos de **DL** suelen dar una **precisión muy por encima de los modelos tradicionales**. Además, exigen un **menor procesamiento de datos**.

En otros casos también son usados modelos de DL, pero hay varios puntos importantes:

- Suelen ser menos interpretables.
- Las empresas que empiezan con IA necesitan coger confianza con este campo y el DL suele definirse como una black-box.
- No siempre se dispone de GPUs.

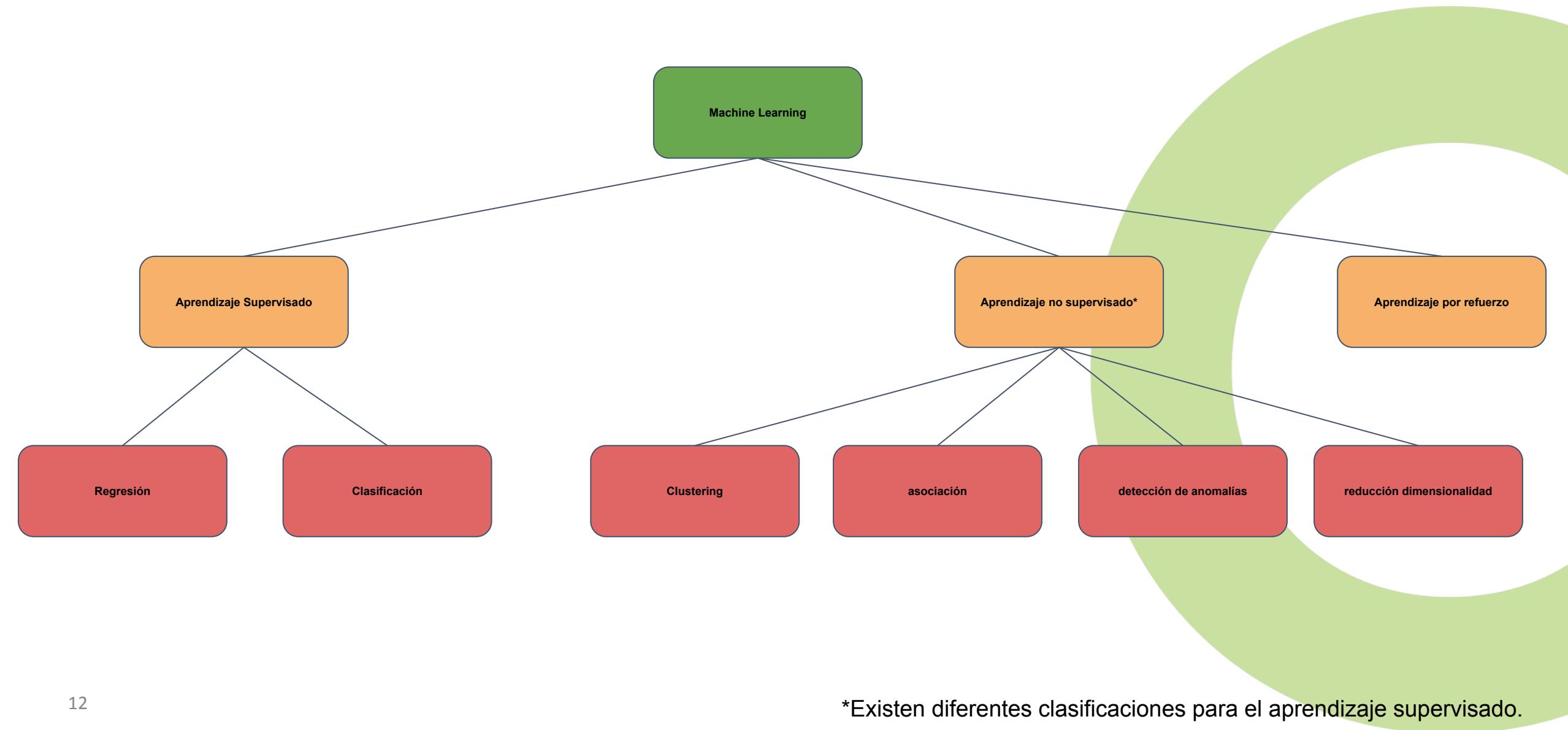


Índice de la sesión

- **Recap**
 - Introducción
 - **Árboles de decisión y ensembles**
 - Hyperparameter tuning
 - Explicabilidad
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- Modelos de Deep Learning

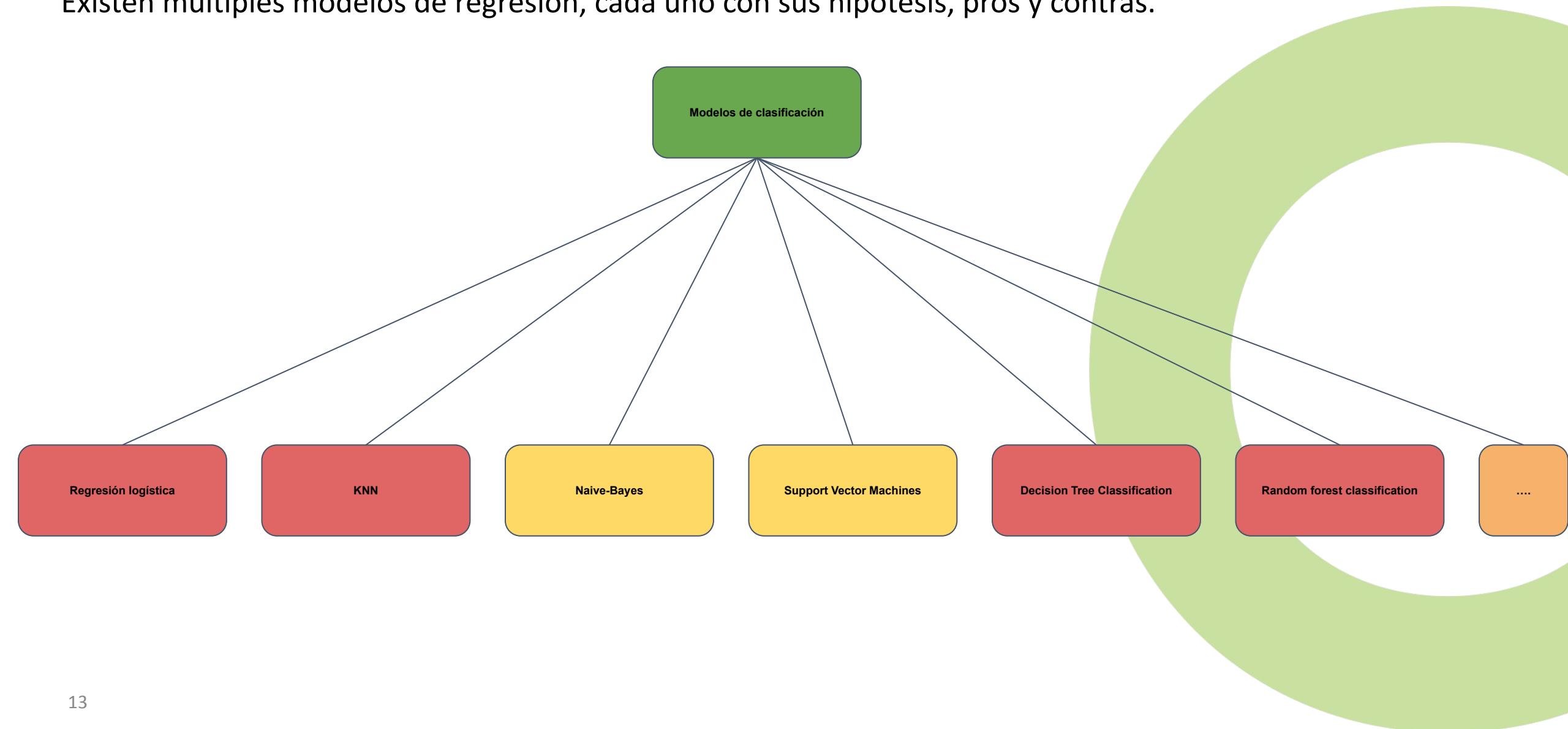


Tipos de Modelos de Machine Learning



Modelos de clasificación

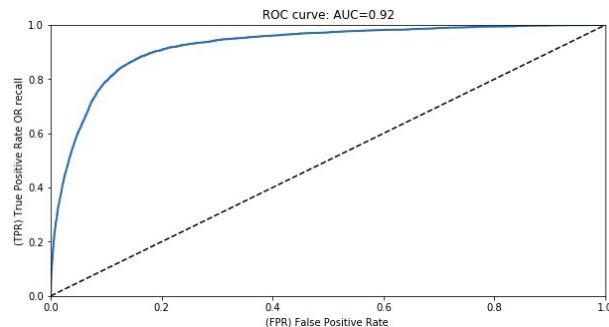
Existen múltiples modelos de regresión, cada uno con sus hipótesis, pros y contras.



Métricas de modelos de clasificación

Los modelos de clasificación tienen una serie de métricas que permiten medir el performance del mismo.

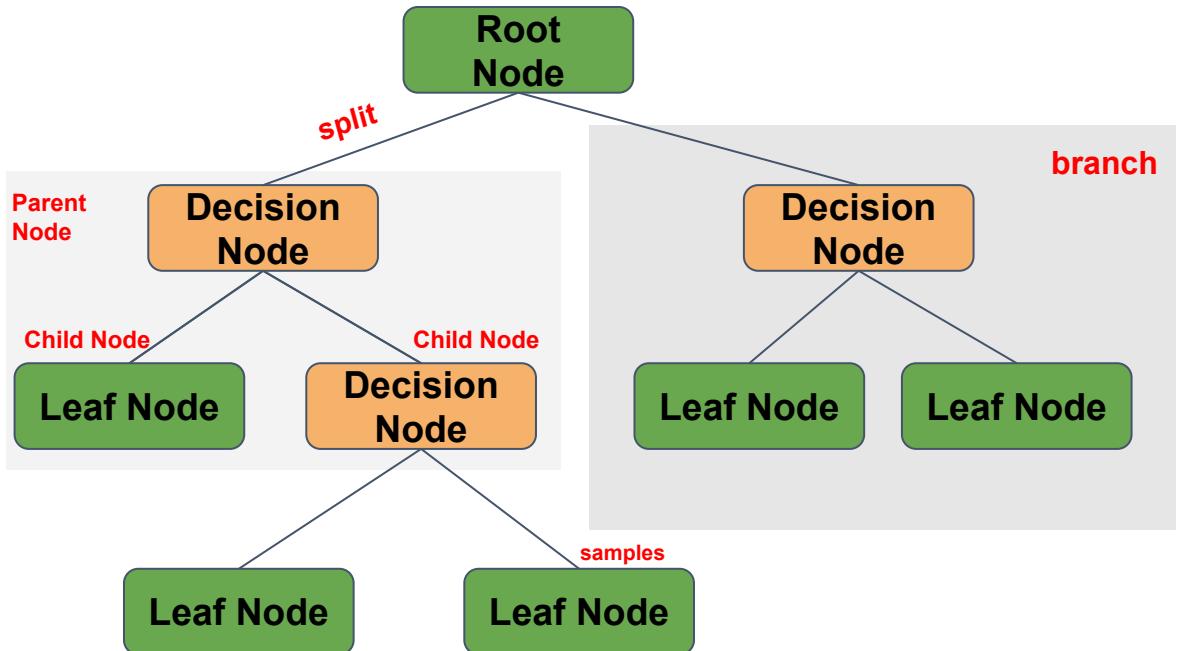
- **Accuracy:** Mide la magnitud promedio de los errores en el set de predicciones, sin considerar la dirección de los mismos.
- **Precision:** De todos los predichos como real por el modelo, cuántos son ciertos.
- **Recall:** De todos los datos reales, cuántos son correctamente predichos por el modelo.
- **F1-Score:** Combina Precisión y Recall en una única métrica.
- **AUC:** Área debajo de la curva.
 - ROC: Curva TPR-FPR.
 - PR AUC: Curva Precisión-Recall.



		Pred	
		TP	
Real	FP	TN	
		Predicted Class	
Actual Class		Positive	Negative
Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP + FN)}$
Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
	Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

Árboles de decisión

Definamos unos conceptos previos para ver un caso de uso con un árbol de decisión.



Nombre de los nodos en función de su parentesco

Parent Node: Respecto a un nodo, su precedente.
Child Node: Nodo posterior a un nodo padre.

Nombre de los nodos en función de la posición en el árbol

Root Node: Nodo principal de decisión. Primer nodo del árbol.
Decision Node: Nodo intermedio del árbol.
Leaf Node (Terminal Node): Nodo terminal del árbol.

Otros conceptos importantes

Split: División de un nodo Padre.
branch: rama con múltiples nodos.
Depth: Profundidad de una rama o árbol.
Samples: Número de muestras contenidas en un nodo.

Ensembles

Para mejorar los resultados predictivos y estabilidad de los *weak classifiers*, se usan los random forest, que son conjuntos de árboles de decisión cada uno entrenado con una muestra ligeramente distinta de los datos de entrenamiento generada mediante bootstrapping.

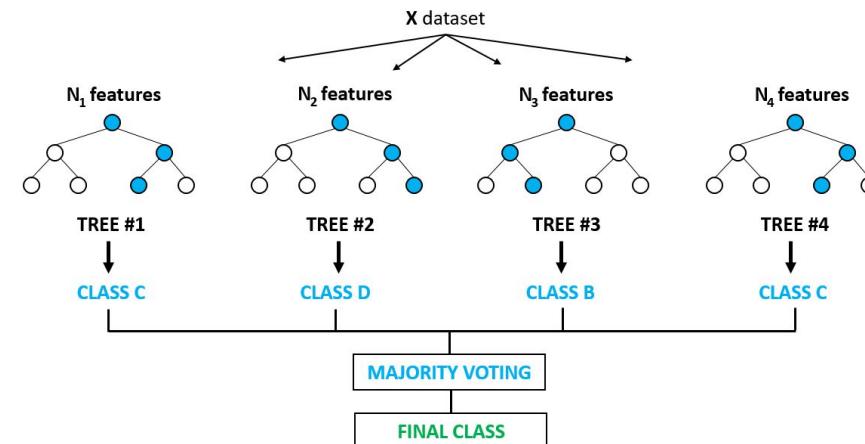
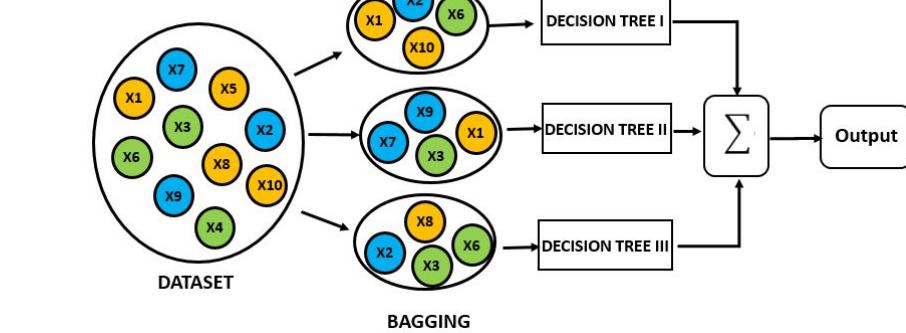
1. Se ejecutan K árboles de regresión diferentes. Para cada árbol:

1.a Se coge un subset de datos aleatorios con los que se va a entrenar el árbol (S_r) y un subset de variables D.

1.b Se crea un árbol de decisión completo (sin hacer pruning) a partir de los datos S_r y variables D.

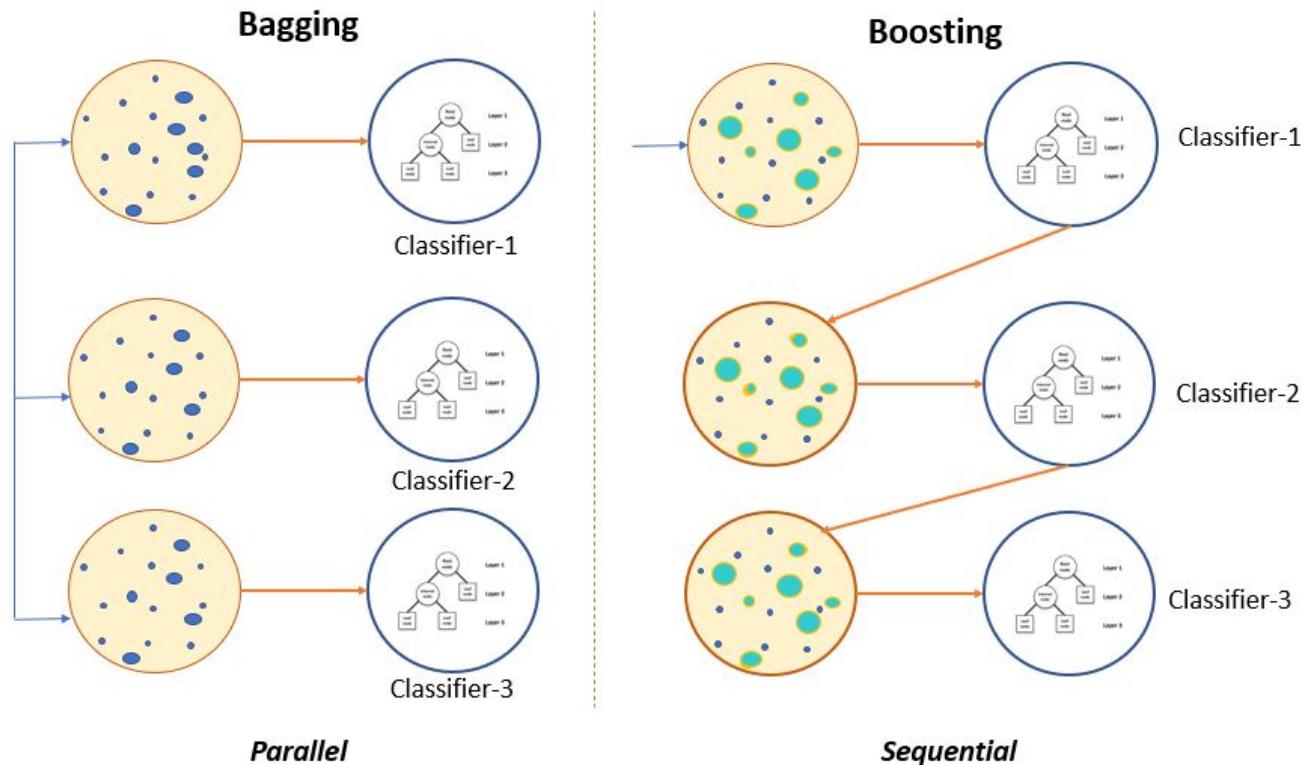
1.c Se calcula el gain de cada árbol.

2. Para realizar predicciones, se pasa por cada árbol los datos de la nueva observación. Cuando tenemos la salida de todos esos árboles, usamos un sistema de votación mayoritario para decidir la predicción: se elige la clase más predicha por los árboles.



Ensembles

Los modelos de Random Forest son un ensemble tipo bagging, debido a que se lanzan K modelos independientes ejecutándose en paralelo. Sin embargo, existen otros modelos denominados tipo boosting.

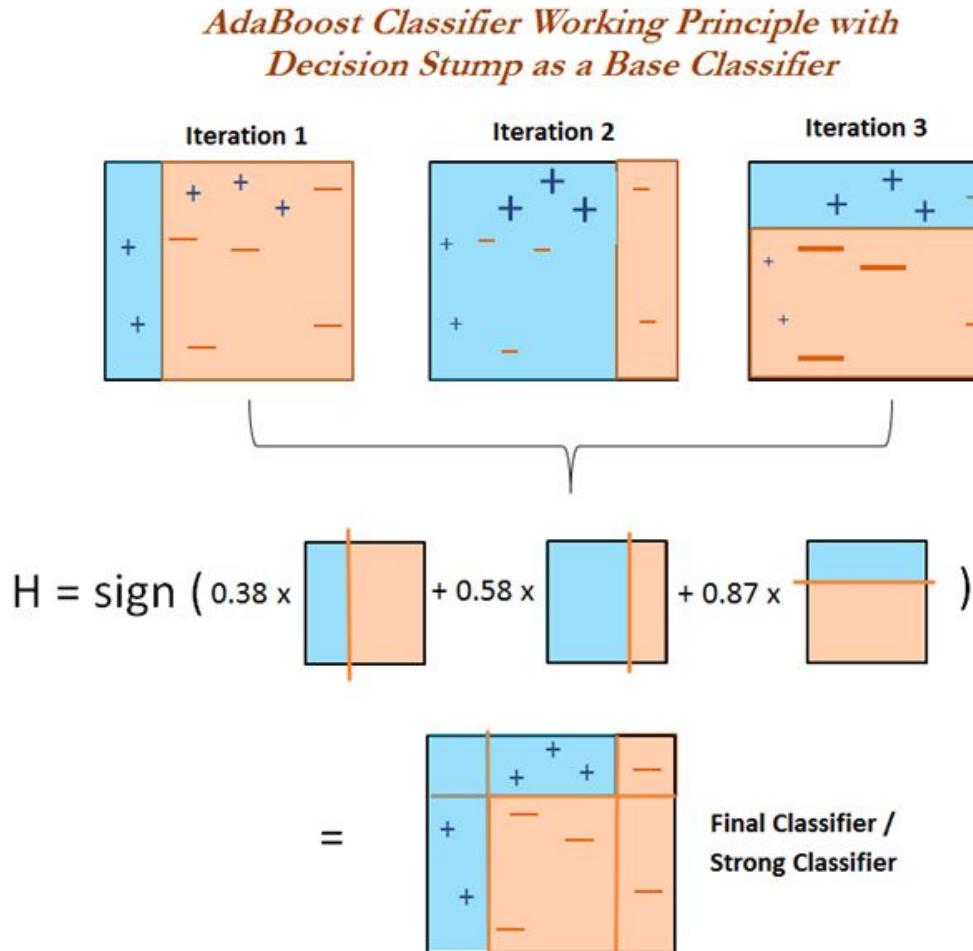


Se calculan múltiples modelos paralelamente e independientemente entre ellos. A posteriori se realiza una predicción basada en un sistema de votación mayoritario.

Los modelos se calculan de manera secuencial. El output y error de cada modelo es utilizado en el siguiente modelo para mejorar la predicción.

Ensembles - AdaBoost

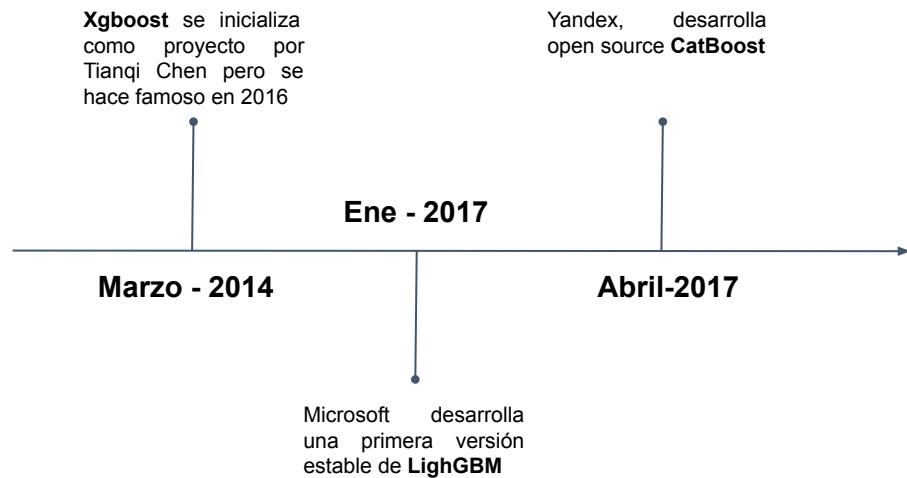
Un ejemplo de cómo funcionan algunos modelos de boosting puede ser el Adaboost.



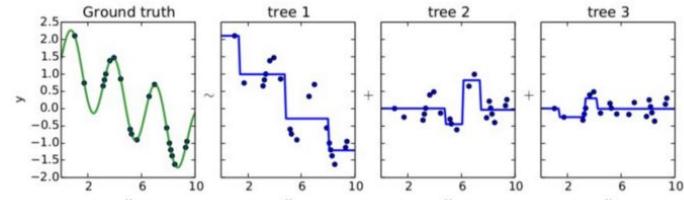
1. Se realiza un primer *weak classifier* y se evalúa qué puntos han sido correctamente e incorrectamente clasificados.
2. A los puntos mal asignados se les da más peso y se intenta hacer otro *weak classifier* que los clasifique correctamente.
3. Se vuelve a realizar el mismo proceso con los puntos mal clasificados.
4. Al finalizar el proceso, se calcula la predicción final mediante la suma ponderada de los *weak classifiers*.

Ensembles

Existen otros modelos basados en árboles basados en gradient boosting: XGboost, LightGBM o CatBoost.



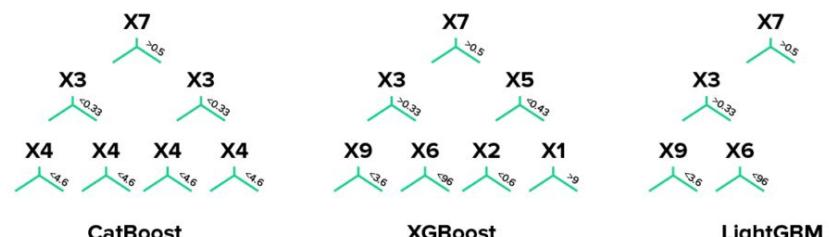
A diferencia del Adaboost, los modelos de **Gradient Boosting** se centran en mejorar las predicciones a partir de los **residuos computados en las iteraciones previas** en vez de dar un mayor peso a las observaciones incorrectamente clasificadas.



Ejemplos de modelos de tipo Boosting son el **XGboost** el **LightGBM** y **CatBoost**. Estos modelos pueden ser utilizados con GPUs y que el entrenamiento se realice de manera paralelizada.

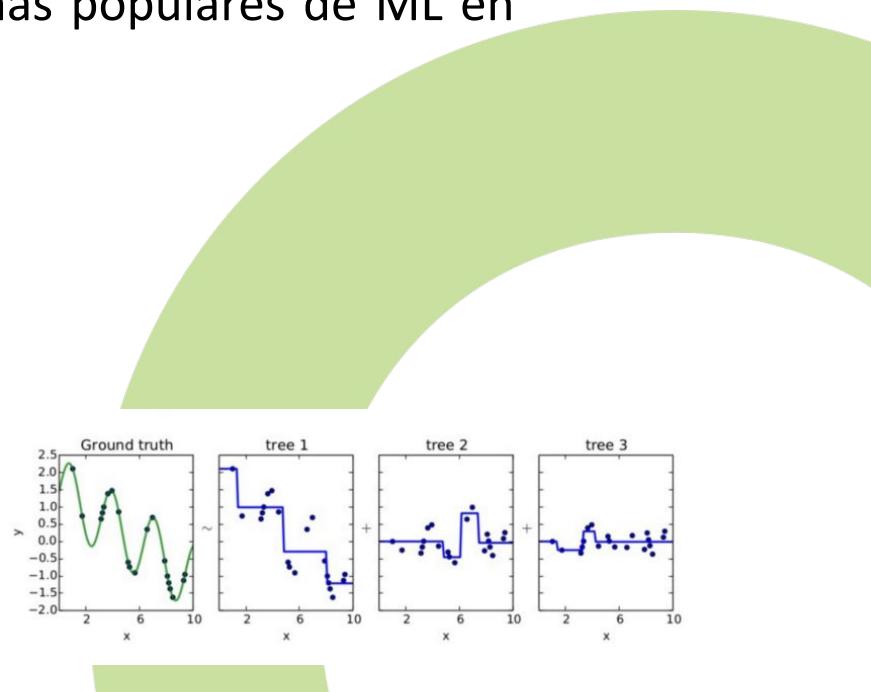
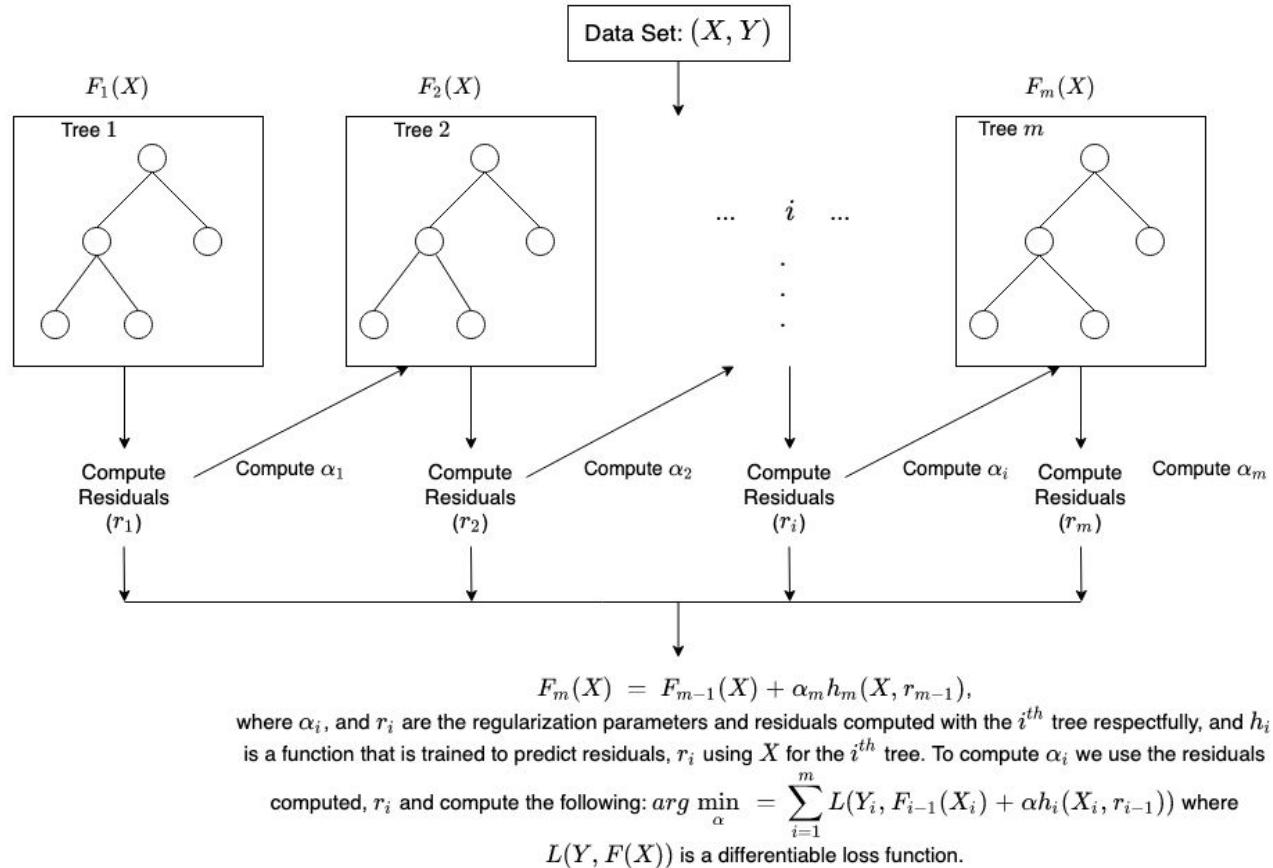
- **XGboost**
 - Método level-wise de crecimiento.
 - Función de pérdida regularizada para evitar sobreajuste.
 - Es necesario un preprocessamiento de las variables categóricas (p.e. OHE).
- **Catboost**
 - Método level-wise de crecimiento.
 - Esquema de partición aleatoria y simétrica durante el entrenamiento para evitar sobreajuste.
 - Optimizado para variables categóricas, no es necesario un preprocessamiento.
- **LightGBM**
 - Método leaf-wise de crecimiento.
 - Veloz y escalable para datos con gran número de observaciones y features.

Tree growth examples:



Ensembles - XGboost

El eXtreme Gradient Boosting Trees (XGBoost) es uno de los modelos más populares de ML en casos de uso tabulares.



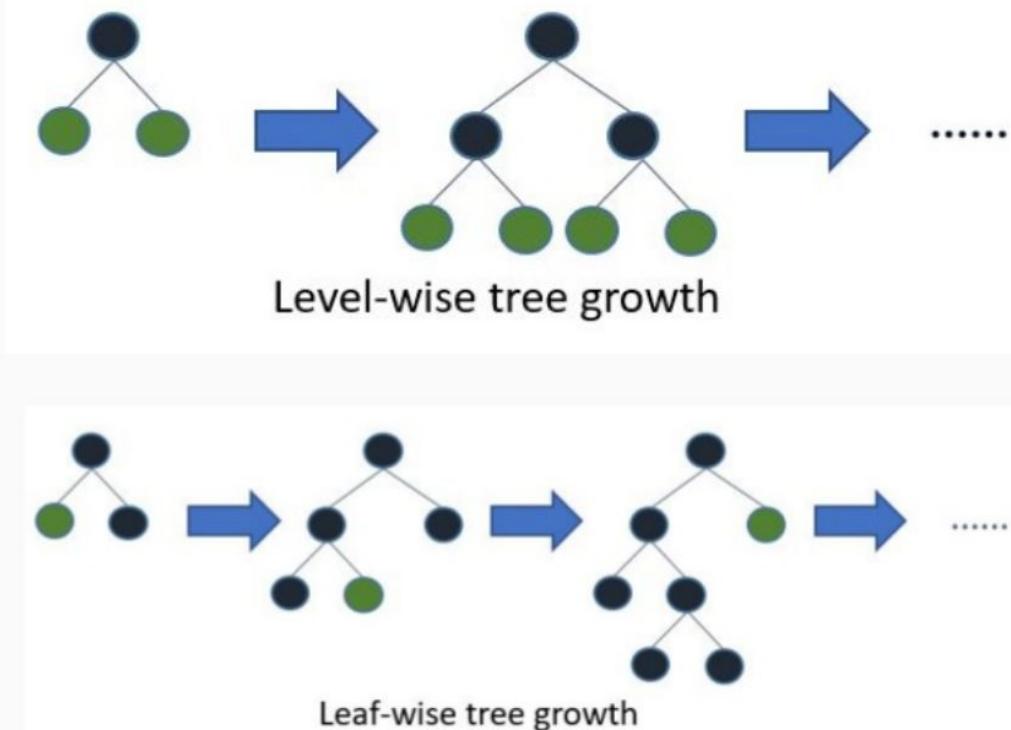
Ensembles - lightGBM

Diferencia entre los métodos level-wise y leaf-wise.

XGBoost

Catboost

LightGBM

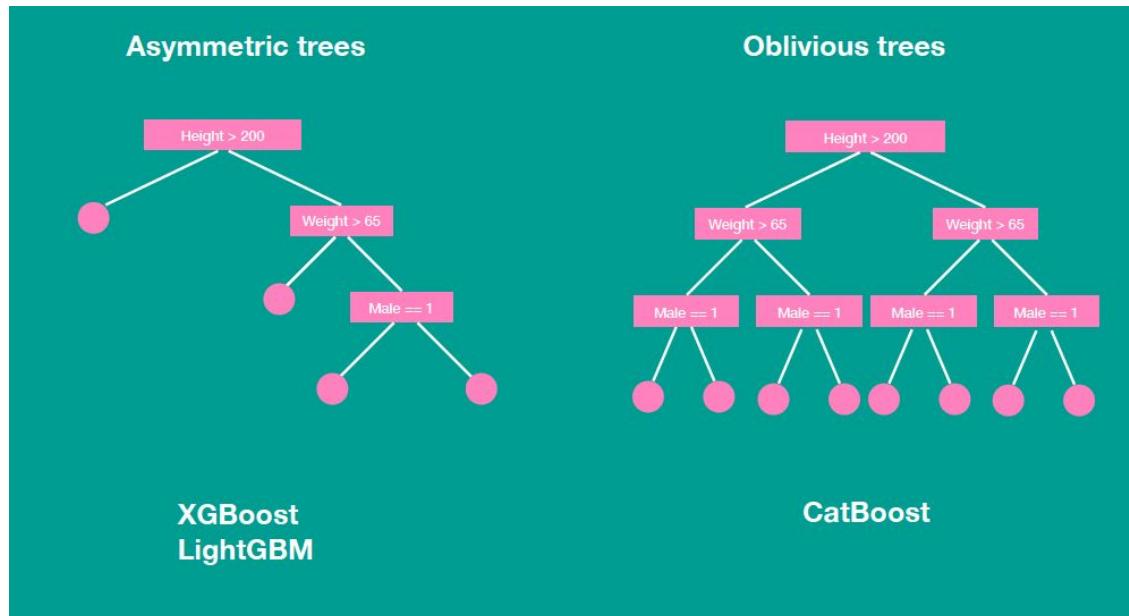
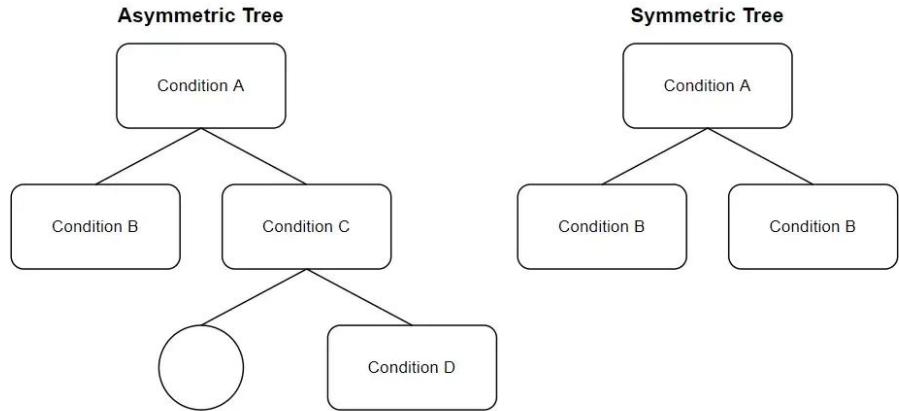


- El árbol crece por niveles, lo que significa que se divide cada nodo en un nivel específico antes de pasar al siguiente nivel
- Puede llevar a sobreajuste

- El árbol crece por hojas, lo que significa que se divide el nodo que produce la mayor reducción en la función de pérdida en lugar de expandir todos los nodos de un nivel específico.
- Este enfoque puede resultar en árboles más profundos y desequilibrados, pero puede ser más eficiente en términos de reducción de error, ya que se enfoca en las divisiones que tienen el mayor impacto en la función de pérdida.

Ensembles - Catboost

El Catboost tiene un sistema level-wise y tiene un sistema de optimización de variables categóricas.



- Catboost utiliza el Mean Target Encoding para tratar las variables categóricas
- Es algo más lento que XGboost o LightGBM por ello.

Ensembles - Catboost

El mean target encoding permite codificar las variables categóricas en función del valor de la variable objetivo.

1. Se dividen los datos de entrenamiento en fold_count (por defecto 10).

1.a Se calcula el promedio de la variable objetivo para cada valor único de la variable categórica

1.b Los promedios se actualizan de manera incremental (en función de las estimaciones)

1.c Para evitar sobreajuste, CatBoost utiliza un esquema de regularización en función de la frecuencia de cada valor en la variable categórica.

2. En el conjunto de datos de validación o prueba, se utiliza el promedio global del objetivo en función de cada valor único de la característica categórica calculado en los datos de entrenamiento para realizar la transformación.

Numerical value	Animal	Good
1.5	cat	0
3.6	cat	1
42	dog	1
7.1	crocodile	1



Ensembles

Si queremos hacer uno de estos modelos con Python, las librerías más usadas con las siguientes.

Algoritmo	Librería Principal	Notebooks Ejemplo
Decision Tree	Scikit-learn sklearn.tree.DecisionTreeRegressor() sklearn.tree.DecisionTreeClassifier()	Iris Diabetes
Random Forest	Scikit-learn sklearn.ensemble.RandomForestClassifier()	Iris Diabetes
XGBoost	XGBoost xgboost.XGBRegressor() xgboost.XGBClassifier()	Iris Diabetes
CatBoost	Catboost catboost.CatBoostRegressor() catboost.CatBoostClassifier()	Iris Diabetes
LightGBM	LightGBM lightgbm.LGBMRegressor() lightgbm.LGBMClassifier()	Iris Diabetes

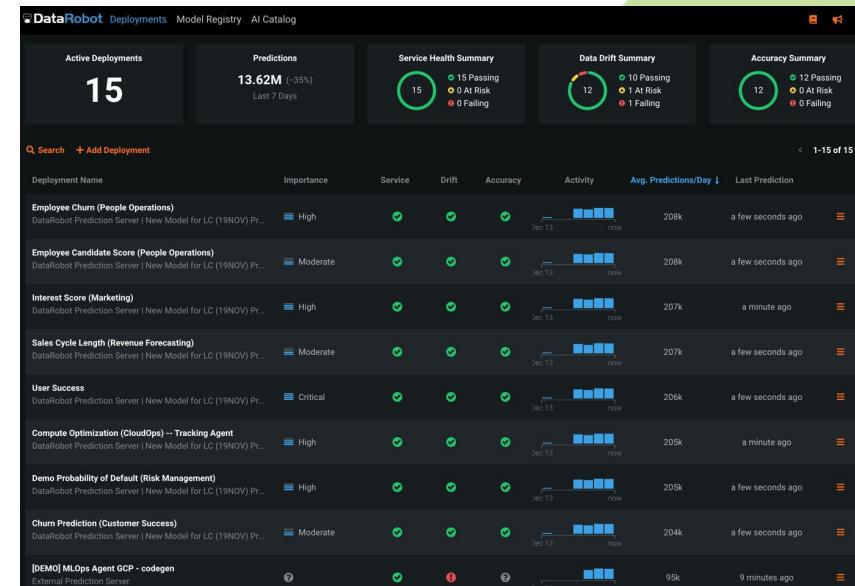
Selección de Modelos

Existen herramientas que permiten probar múltiples modelos con diferentes hiperparámetros.

Python

Model	Accuracy	AUC	Recall	Prec.	F1	Kappa
0 CatBoost Classifier	0.8191	0.881	0.7298	0.8151	0.77	0.6217
1 Gradient Boosting Classifier	0.8188	0.8823	0.7316	0.8133	0.7701	0.6213
2 Extreme Gradient Boosting	0.8168	0.882	0.7202	0.8165	0.7652	0.6161
3 Ada Boost Classifier	0.7961	0.8599	0.7095	0.779	0.7424	0.5743
4 Extra Trees Classifier	0.794	0.8581	0.6768	0.7961	0.7315	0.5661
5 Random Forest Classifier	0.7517	0.8087	0.5913	0.7566	0.6636	0.4715
6 Decision Tree Classifier	0.7301	0.7231	0.6819	0.6724	0.6767	0.4452
7 Ridge Classifier	0.7262	0	0.6454	0.679	0.6615	0.432
8 Logistic Regression	0.7244	0.7344	0.4903	0.7603	0.5959	0.4012
9 Naive Bayes	0.67	0.674	0.5055	0.6302	0.5578	0.3003
10 Linear Discriminant Analysis	0.6621	0.6965	0.603	0.5913	0.5968	0.3061
11 SVM - Linear Kernel	0.5828	0	0.5731	0.6729	0.4805	0.173
12 Quadratic Discriminant Analysis	0.5811	0.5054	0.0624	0.4438	0.1081	0.0124
13 K Neighbors Classifier	0.5777	0.5704	0.4087	0.4887	0.445	0.1086

AutoML



Librerías: pycaret, auto-sklearn, autoXGboost, H2O.

Herramientas: DataRobot, BigML, Azure Automated ML, AWS AutoPilot, GCP AutoML,...

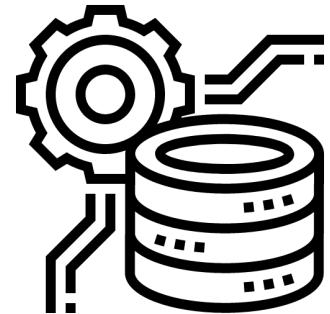
Índice de la sesión

- **Recap**
 - Introducción
 - Árboles de decisión y ensembles
 - **Hyperparameter tuning**
 - Explicabilidad
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- Modelos de Deep Learning

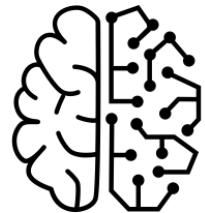


Modelos de Machine Learning

Los modelos de machine learning aprenden patrones de los datos para realizar predicciones sobre datos no vistos anteriormente.



Procesamos los datos



Training

70%



V

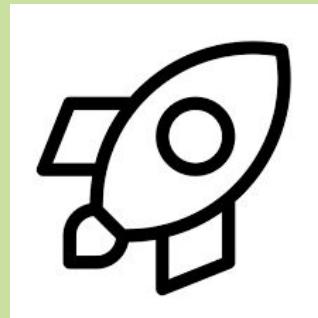
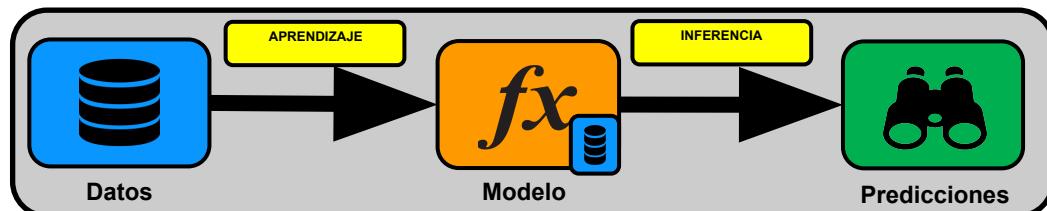
T

20%

10%

Entrenamos el modelo con una parte

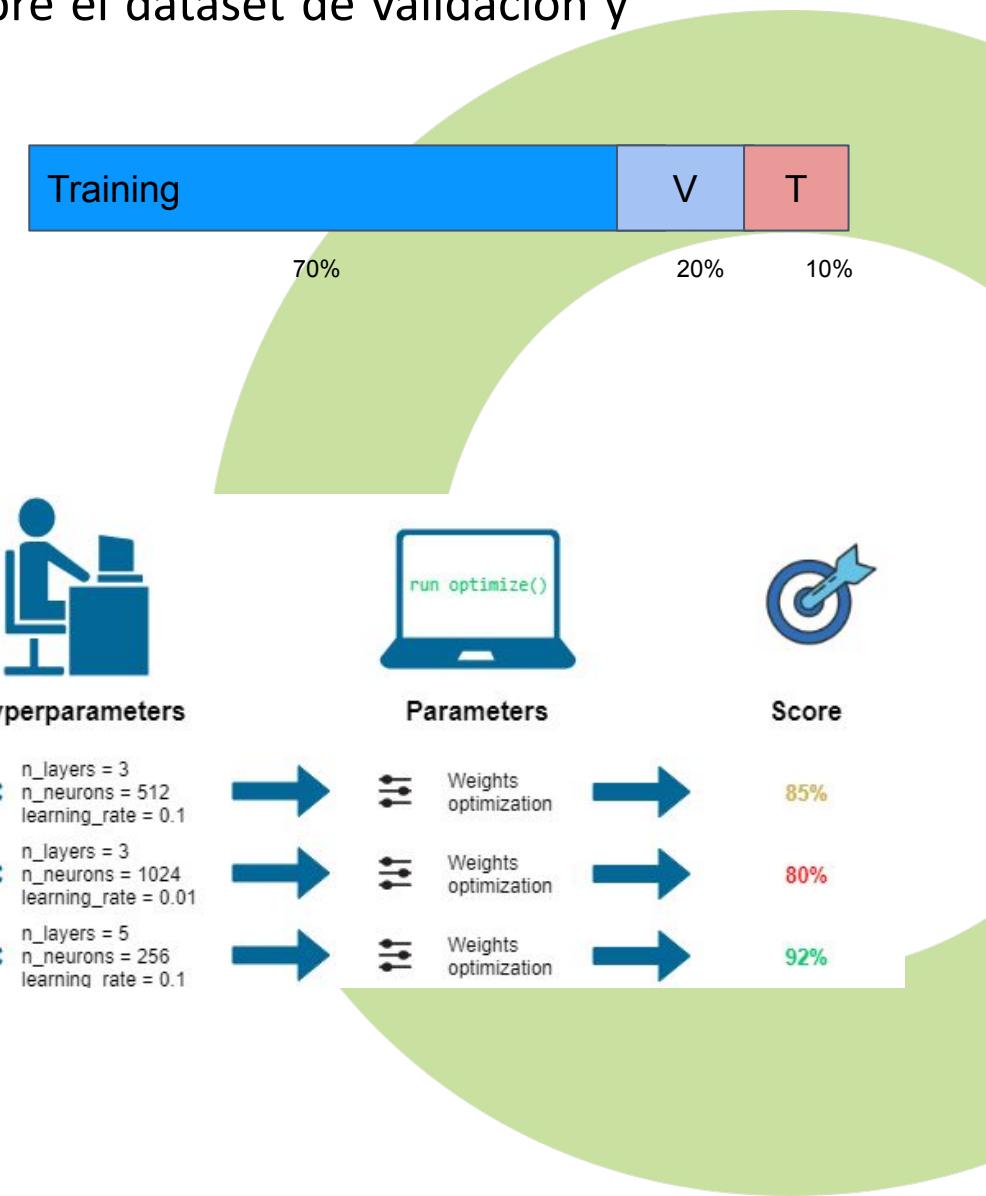
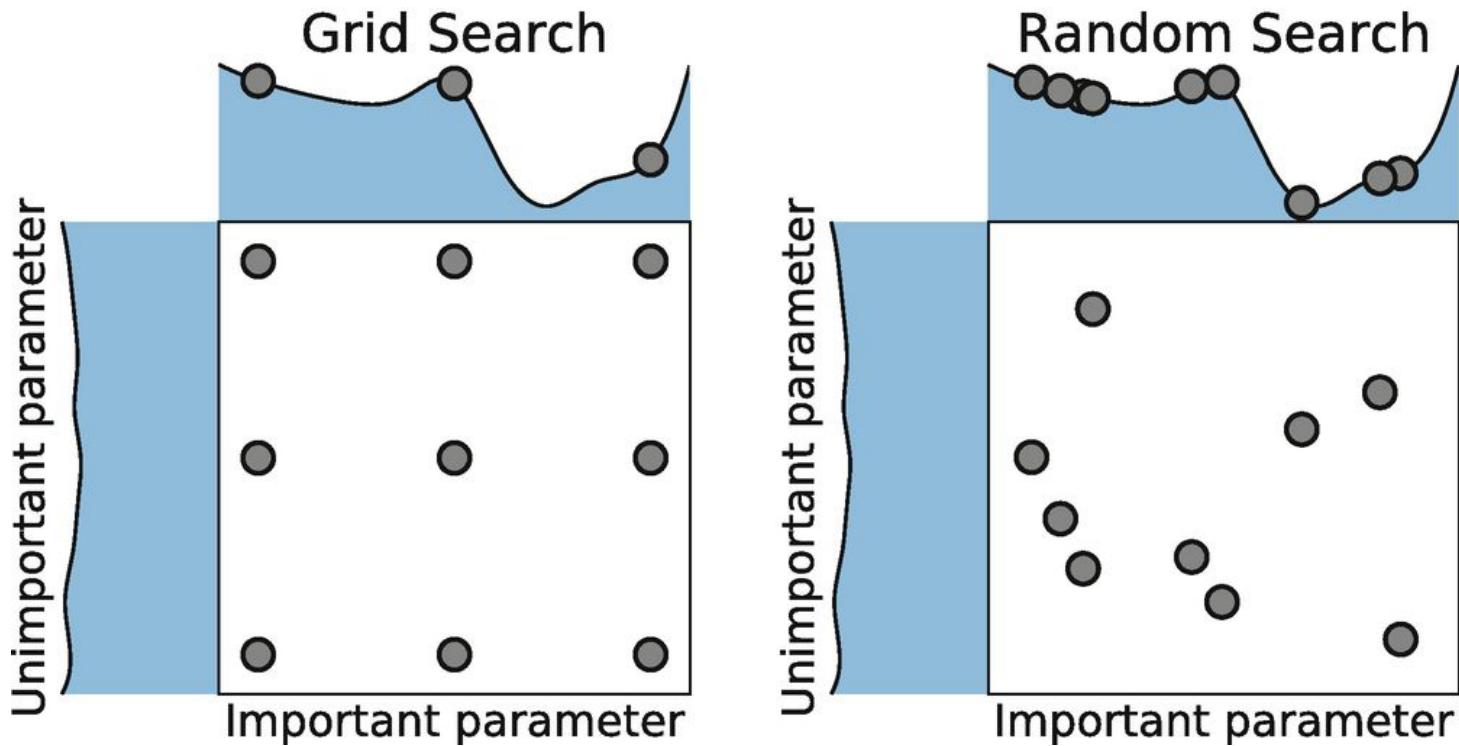
Evaluamos sobre datos que no ha visto el modelo (caso supervisado)



Puesta en producción y predicción sobre nuevos datos

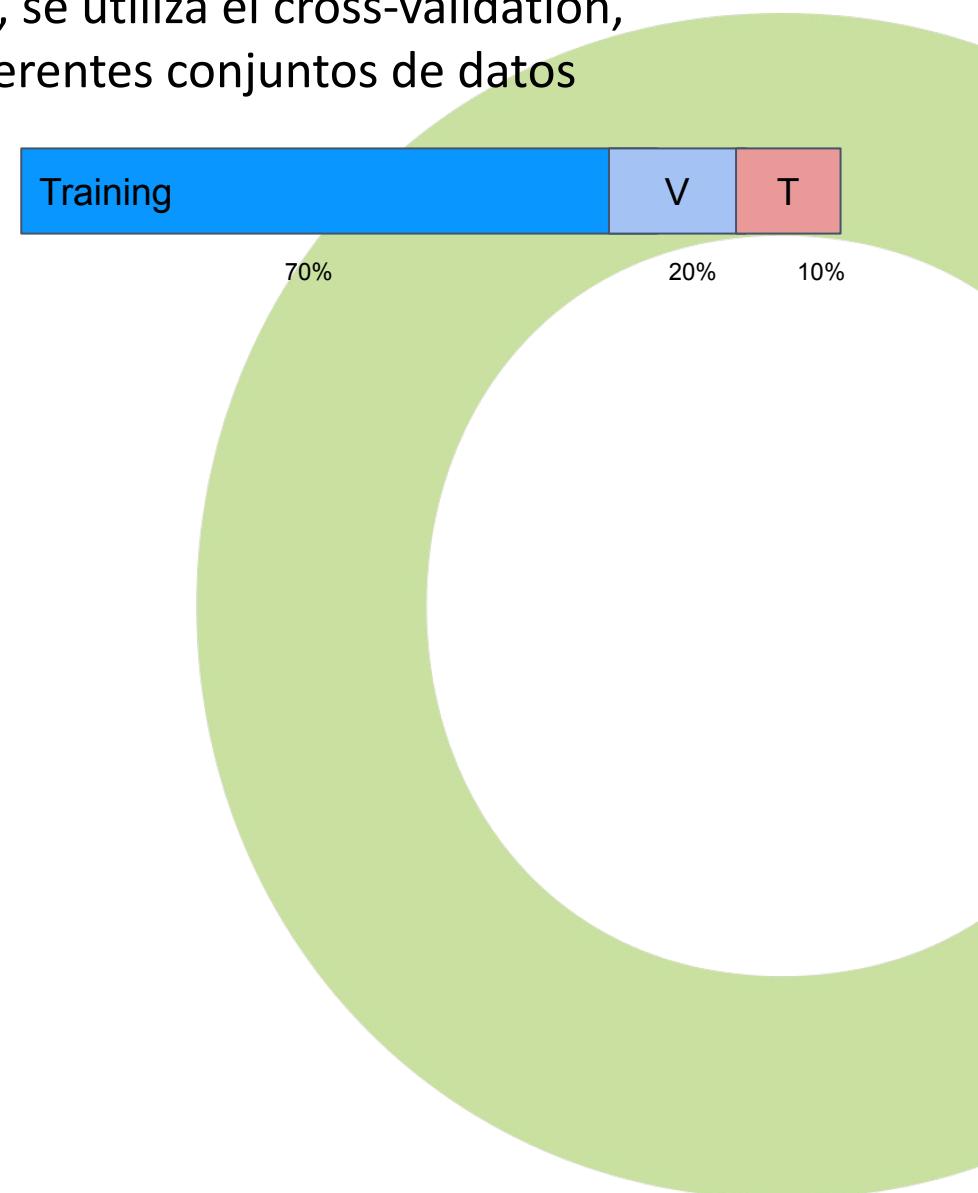
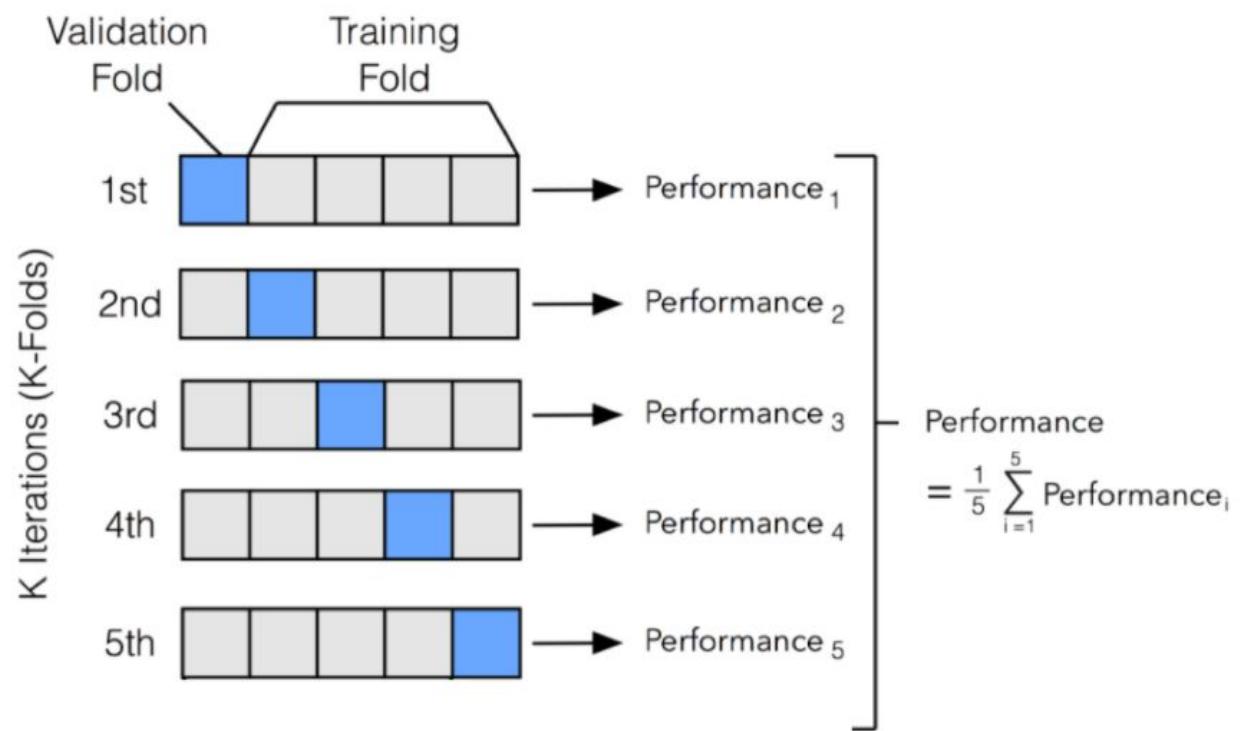
Evaluación de diferentes parámetros

Se van probando diferentes configuraciones de cada parámetro sobre el dataset de validación y se va viendo cual de ellos da mejores métricas.



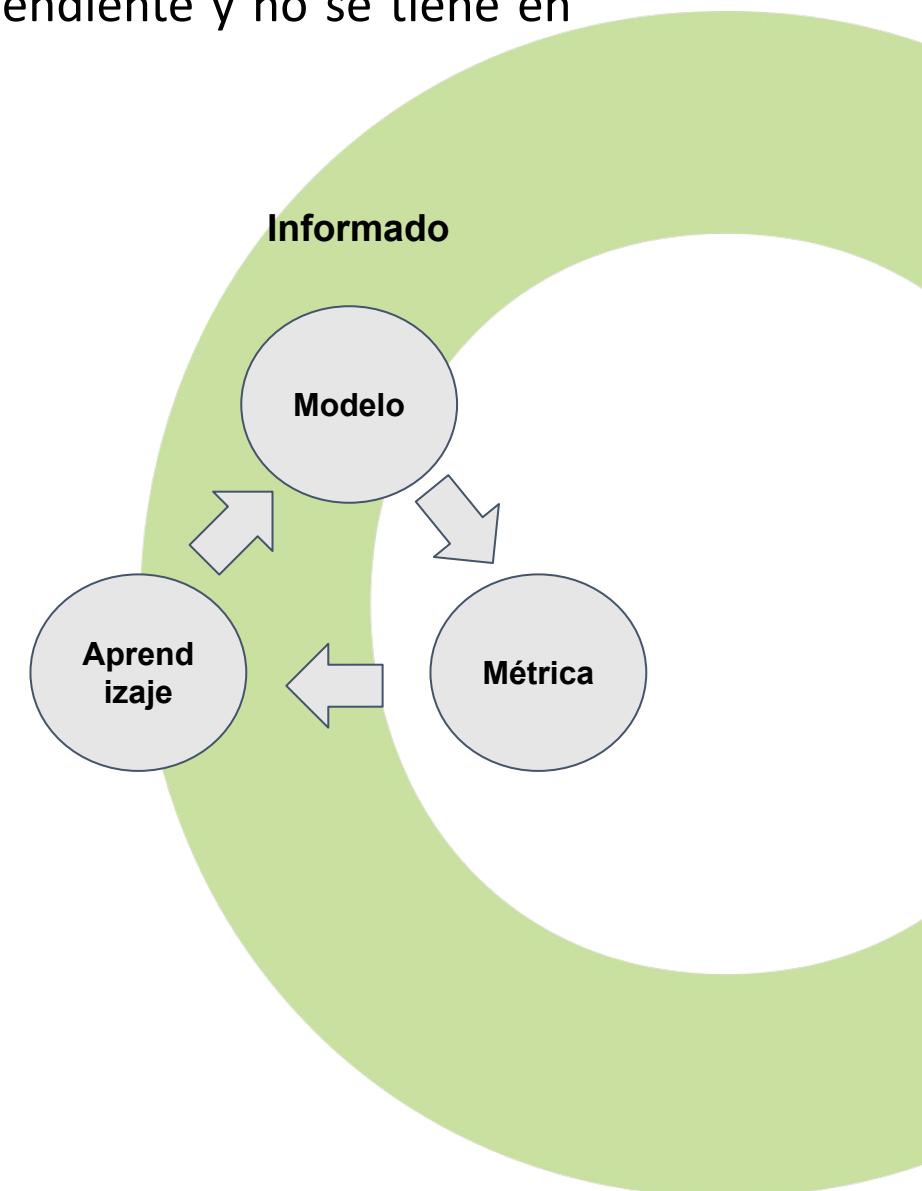
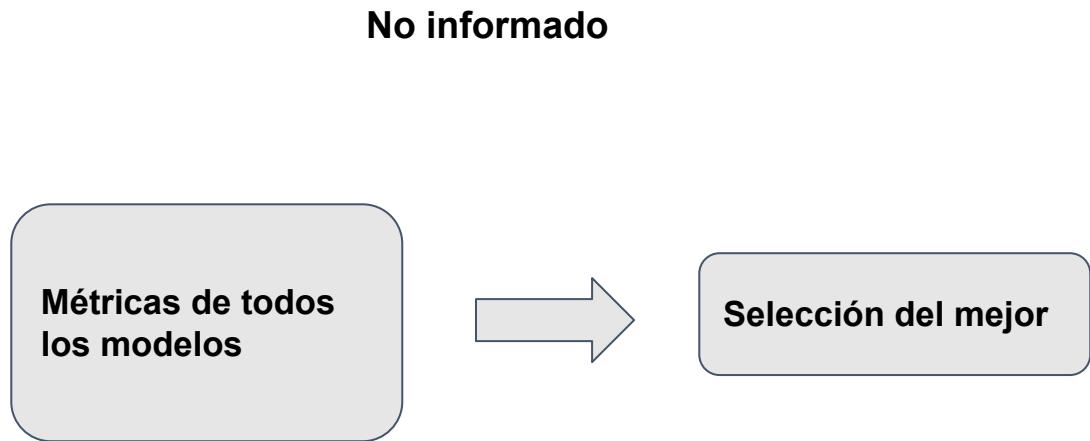
Evaluación de diferentes parámetros

Para hacer que este proceso esté menos sesgado al set de validación, se utiliza el cross-validation, que permite ir evaluando los mejores hiperparámetros utilizando diferentes conjuntos de datos



Problemas comunes al RS y GS

El problema de estos dos algoritmos es que cada evaluación es independiente y no se tiene en cuenta las siguientes para mejorar los resultados.



Índice de la sesión

- **Recap**
 - Introducción
 - Árboles de decisión y ensembles
 - Hyperparameter tuning
 - **Explicabilidad**
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- Modelos de Deep Learning



Diferencia Interpretabilidad & Explicabilidad

Interpretabilidad y Explicabilidad son diferentes conceptos dentro del campo del ML.

Interpretabilidad

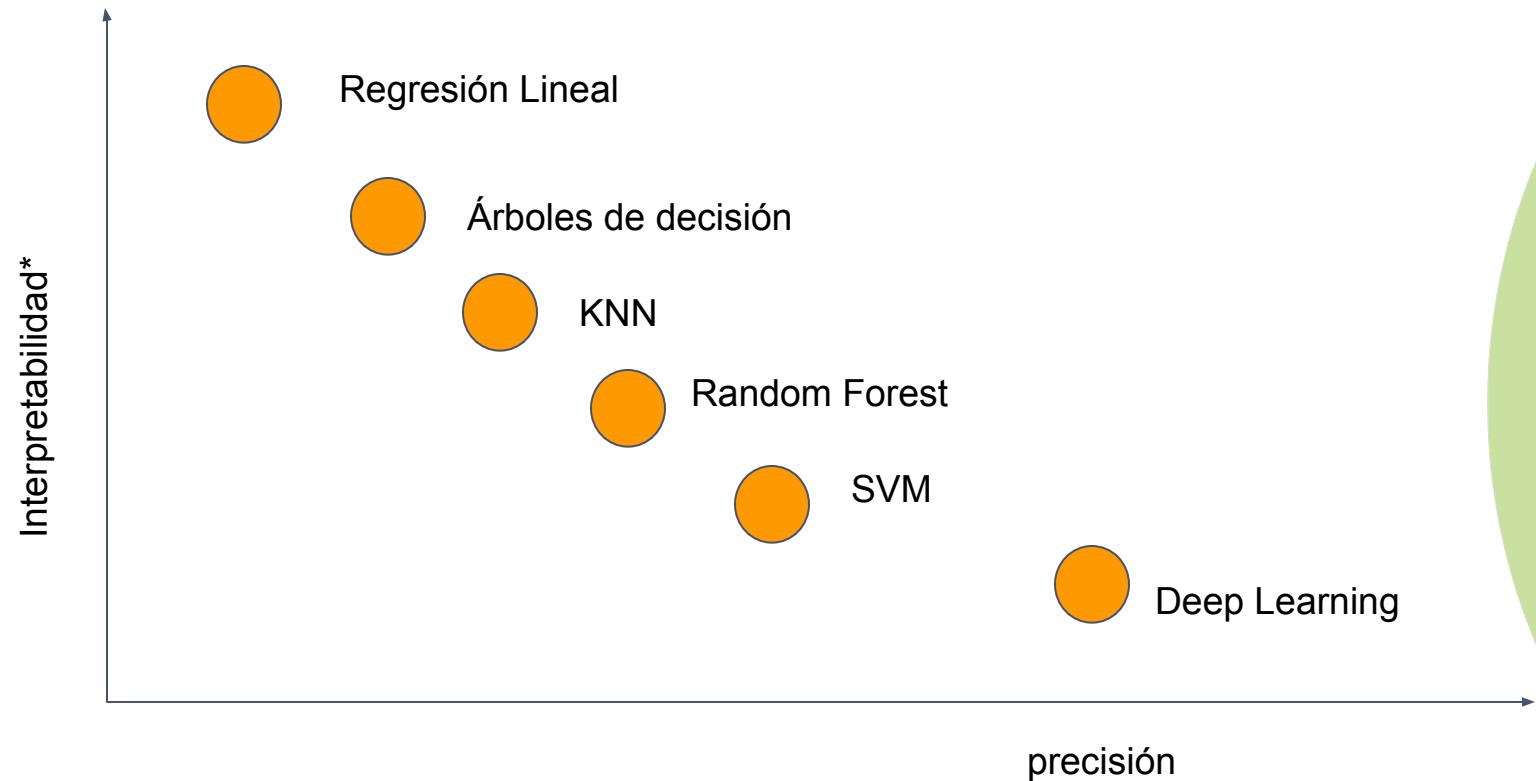
Se refiere a las técnicas que permiten interpretar los modelos de manera inherente, como pueden ser los coeficientes de la regresión lineal, el split de los árboles de decisión o la feature importance.

Explicabilidad

Se refiere al proceso de aplicar métodos que identifican cómo un modelo complejo está tomando las decisiones a partir de técnicas externas al mismo.

Interpretabilidad

En función de la necesidad final del proyecto y del caso de uso, este aspecto es muy importante de definir al principio de un proyecto.



Sin embargo, modelos que tienen una baja interpretabilidad pueden ser explicables como veremos.

Explicabilidad

Se refiere al proceso de aplicar métodos que identifican cómo un modelo complejo está tomando las decisiones a partir de técnicas externas al mismo.

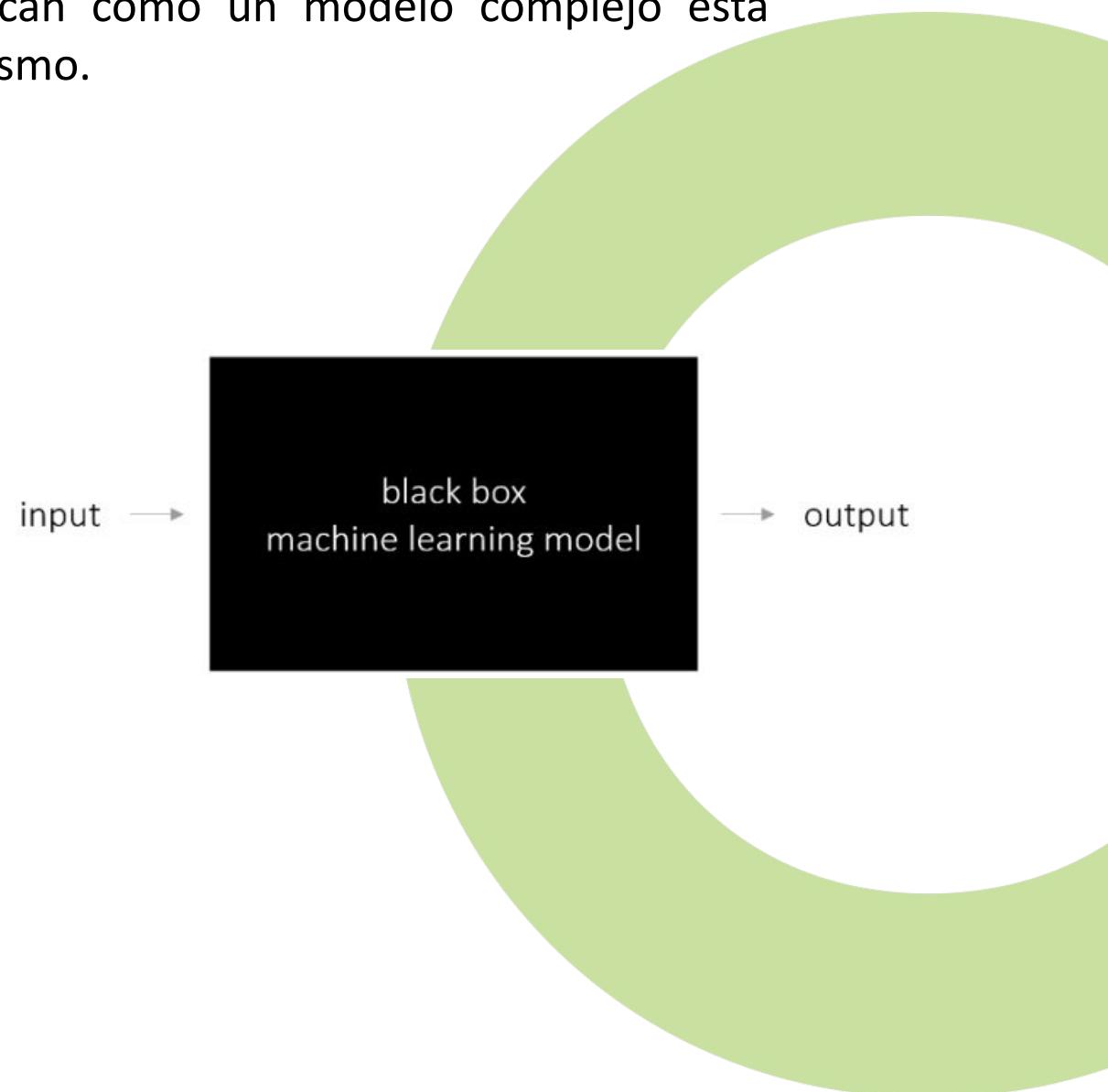
Métodos más conocidos

Globales (describen el comportamiento promedio del modelo):

- Permutation Feature Importance
- Partial Dependence Plots

Locales (explican predicciones individuales):

- Local Surrogate (LIME)
- SHapley Additive exPlanations (SHAP)



Permutation Feature Importance

Permite a partir de un determinado modelo creado, evaluar cuales son las variables más importantes.

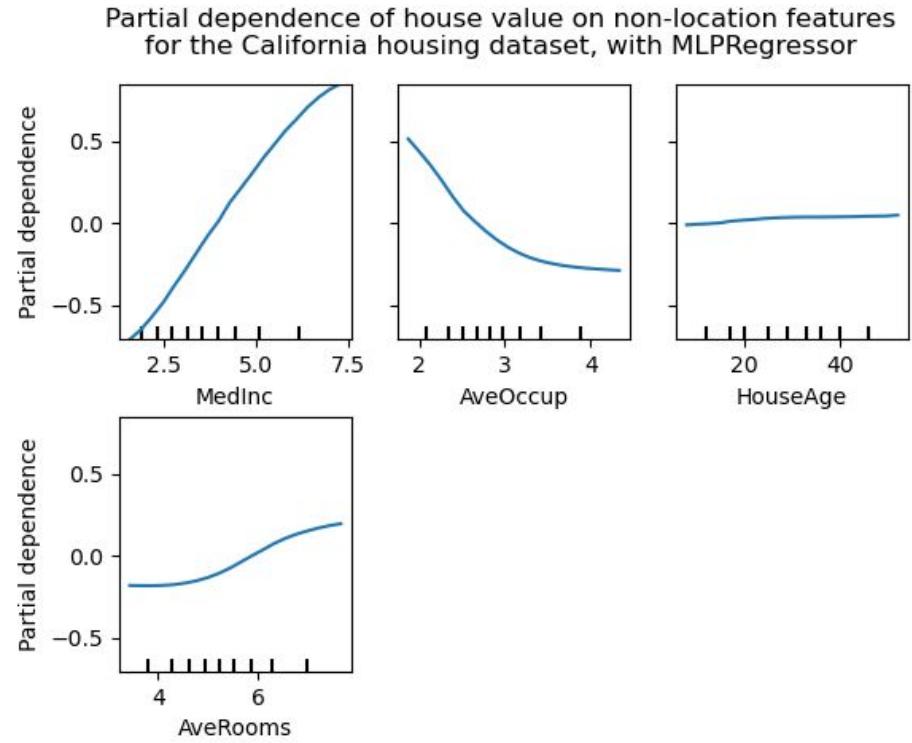
1. Se **permutan los valores** en el set de validación en una determinada variable y se usa el modelo para predecir sobre el set de datos. Eso debería llevar a que empeoraran las métricas ya que esa variable pierde su valor real.
2. La **deterioración del performance** permite evaluar la importancia de esa variable.

Churn	Balance	Nº Transactions	Nº products	Has Loan
0	5000	50	5	1
1	10000	20	2	1
...				
0	3000	10	3	0
1	200	30	1	0

La Permutation importance **es calculada después de que el modelo sea entrenado**, de manera que no cambia el mismo o sus predicciones.

Partial Dependence Plots (PDP)

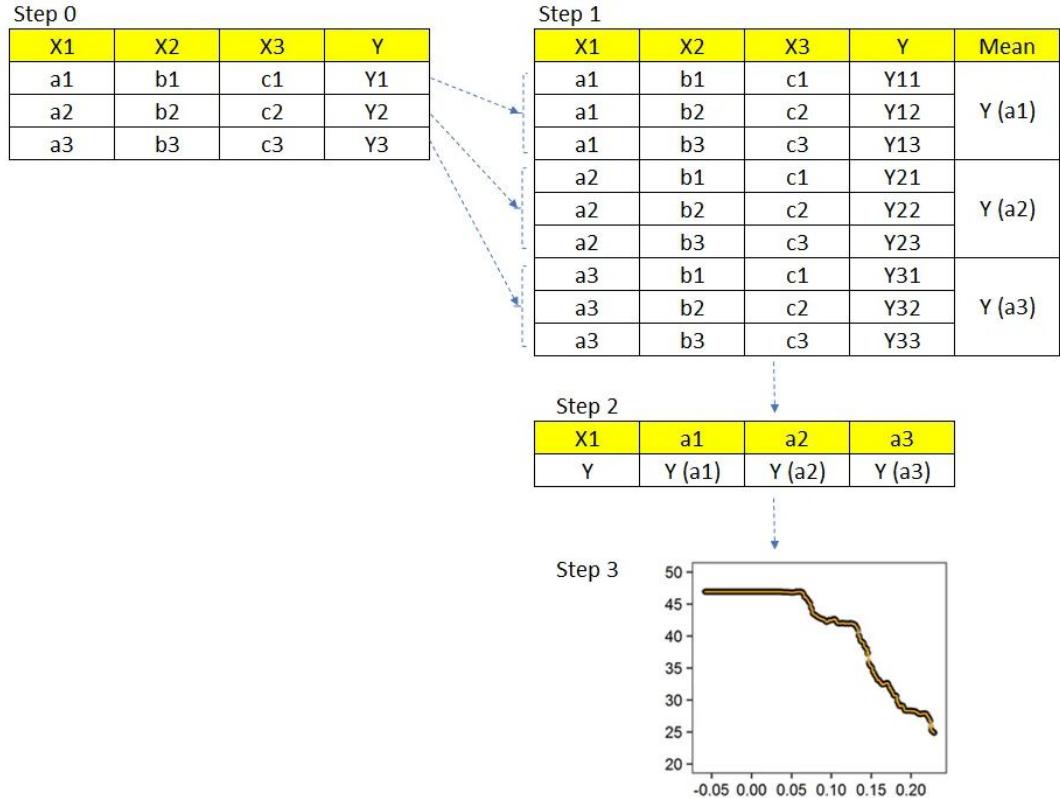
Muestran el efecto marginal que una o dos variables tienen en el output de la predicción.



A partir del PDP podemos ver si una variable tiene una relación lineal, monotónica o más compleja con el target.

Partial Dependence Plots (PDP)

¿Cómo funcionan los PDP?



Partimos de un dataset de 3 variables (X1,X2,X3) y un target Y.

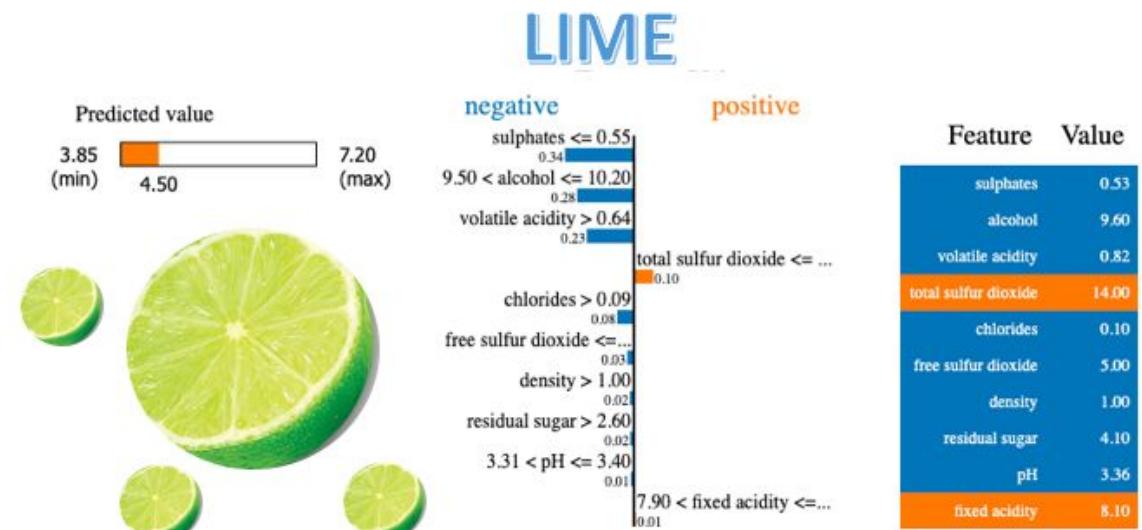
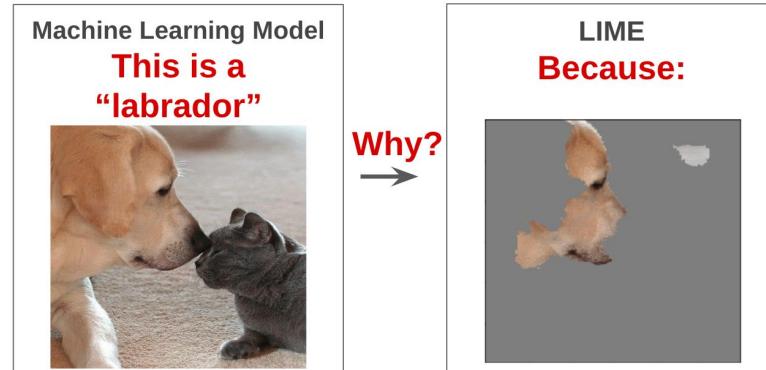
1. Por cada una de las variables del dataset:
 - a. Permutamos cada posible valor de la variable con los demás posibles valores de las demás variables.*
 - b. Generamos una predicción de cada una de las posibles observaciones.
 - c. Promediamos la predicción por cada valor único de la variable.
2. Hacemos un plot de cada valor posible de la variable respecto a los valores promedio de la predicción.

[Más info](#)

Los valores no son imputados, sino que existen ya dentro del dataset.

LIME

LIME explica la toma de decisiones del modelo a nivel de observación (explicabilidad local). Lime evalúa lo que sucede con las predicciones cuando se varían los datos en un modelo de ML interpretable.

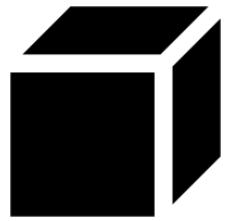


Funciona con datos tabulares, imágenes y texto.

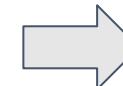
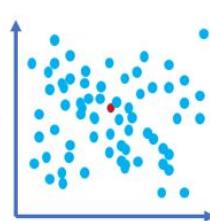
LIME

LIME explica la toma de decisiones del modelo a nivel de observación (explicabilidad local). Lime evalúa lo que sucede con las predicciones cuando se varían los datos en un modelo de ML interpretable.

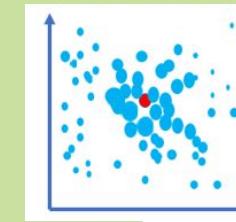
Entrenamos modelo de caja negra



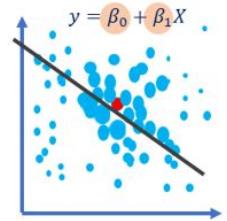
Se generan muestras perturbadas de un punto determinado de interés



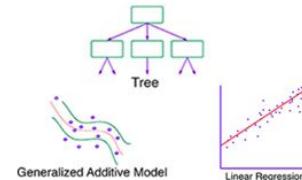
Ponderamos las muestras en función de su distancia a la observación de diferencia



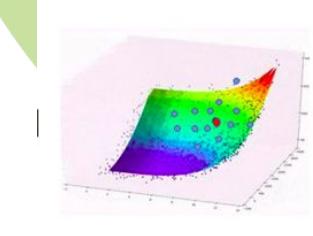
Evaluamos los resultados del modelo explicable



Elegimos un modelo explicable y entrenamos los datos

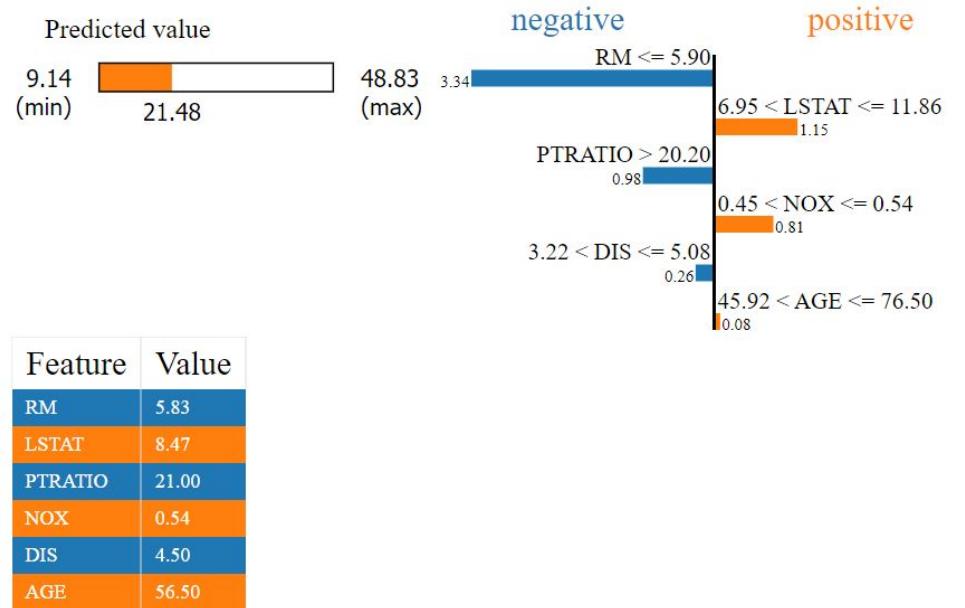


Realizamos predicciones de estos puntos con el modelo de caja negra



LIME

Gráficas con LIME. Ejemplo de un punto.



- **Predicción:** 21.48
- Las **variables LSTAT y NOX** tienen una **influencia positiva** mientras que RM, DIS, AGE y PTRATIO tienen una **influencia negativa**.
- Las **variables que son numéricas** son transformadas a **rangos** como podemos ver en el nombre de las mismas.
- Los números asociados a cada variable indican el peso del modelo creado. P.e. 3.34 para RM.

SHAP Values

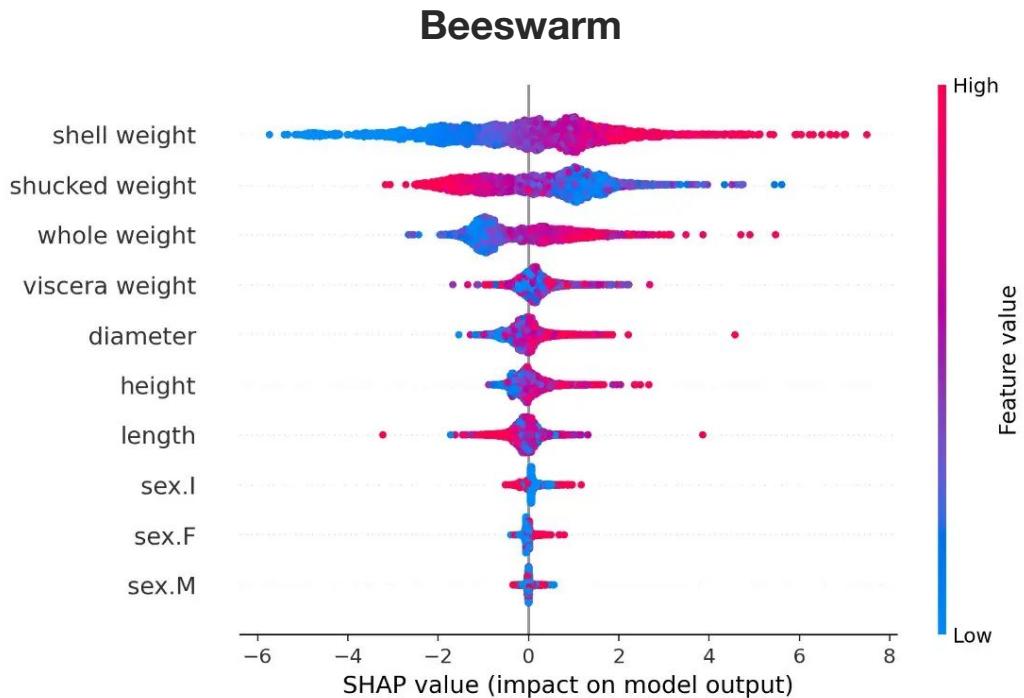
SHAP Values es un método de explicabilidad local que permite evaluar el peso de cada variable en la predicción de un modelo de caja negra.

- Aunque es un método local, se suele agregar los SHAP Values para ver una explicabilidad más genérica del modelo.
- SHAP Values procede de la teoría de juegos de Shapley, que buscan dar una contribución justa a cada uno de los jugadores en juegos de equipo.
- Hay varias visualizaciones para interpretar los SHAP values: waterfall, force, decision, mean SHAP y beeswarm plots.



SHAP Values

Visualizaciones agregadas.



Visualización de todas las observaciones SHAP del dataset agrupadas por features. En esta, vemos de azul a rojo cómo se incrementa el valor de una variable. Esto nos permite ver mejores relaciones.

- Shell weight aumenta los SHAP values a medida que aumenta el valor de la variable. Eso quiere decir que aumenta el número de anillos con la variable shell weight.
- En el caso de shucked weight la relación es inversa.

Índice de la sesión

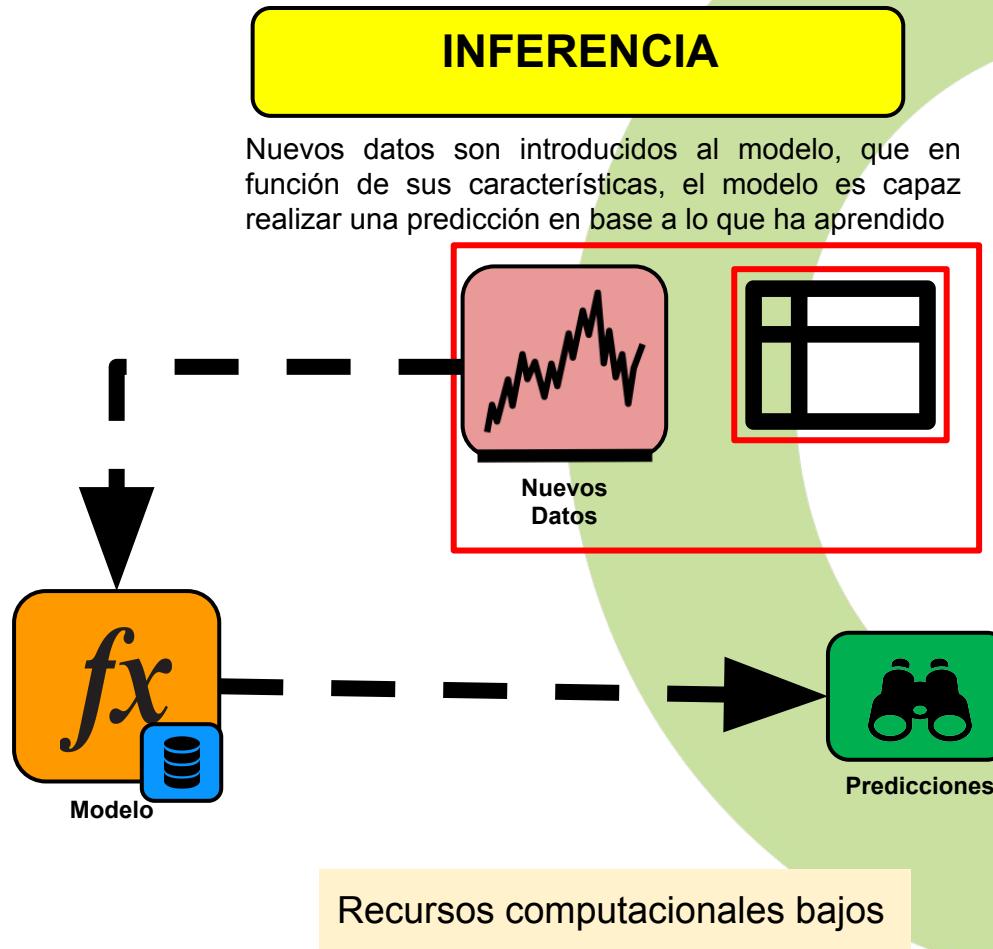
- **Recap**
 - Introducción
 - Árboles de decisión y ensembles
 - Hyperparameter tuning
 - Explicabilidad
 - **Productivización**
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- Modelos de Deep Learning



Batch

Son procesos que se ejecutan periódicamente para hacer predicciones con altos volúmenes de datos.

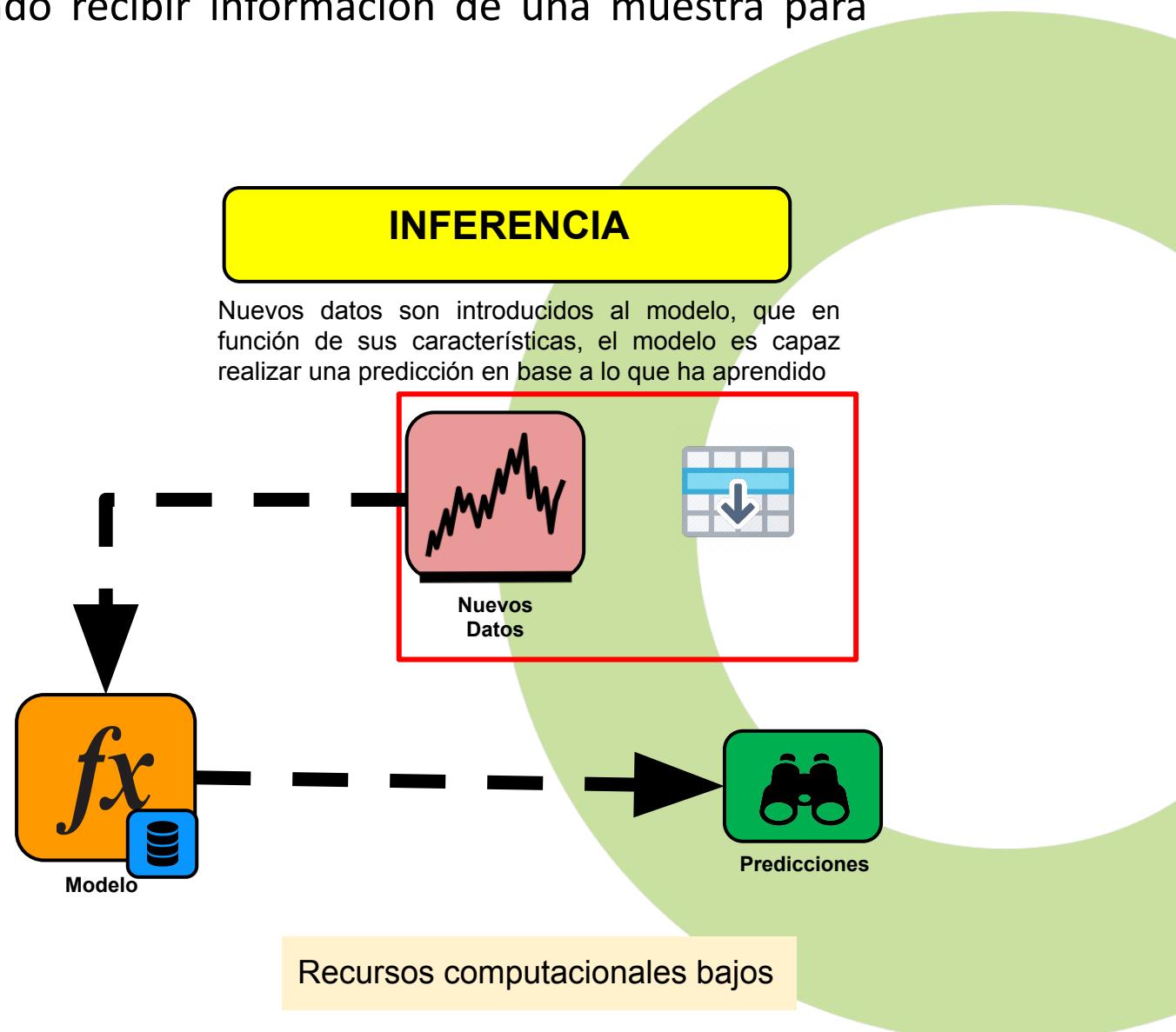
- **Son procesos que se planifican para su ejecución**
 - Por ejemplo cada día, mes, trimestre...
- **Se suele enviar una tabla con muchas muestras sobre las que realizar predicciones.**
 - Muchos datos son pasados al modelo para hacer predicciones.



Real-Time

Son procesos siempre activos, que están esperando recibir información de una muestra para realizar las predicciones de la misma.

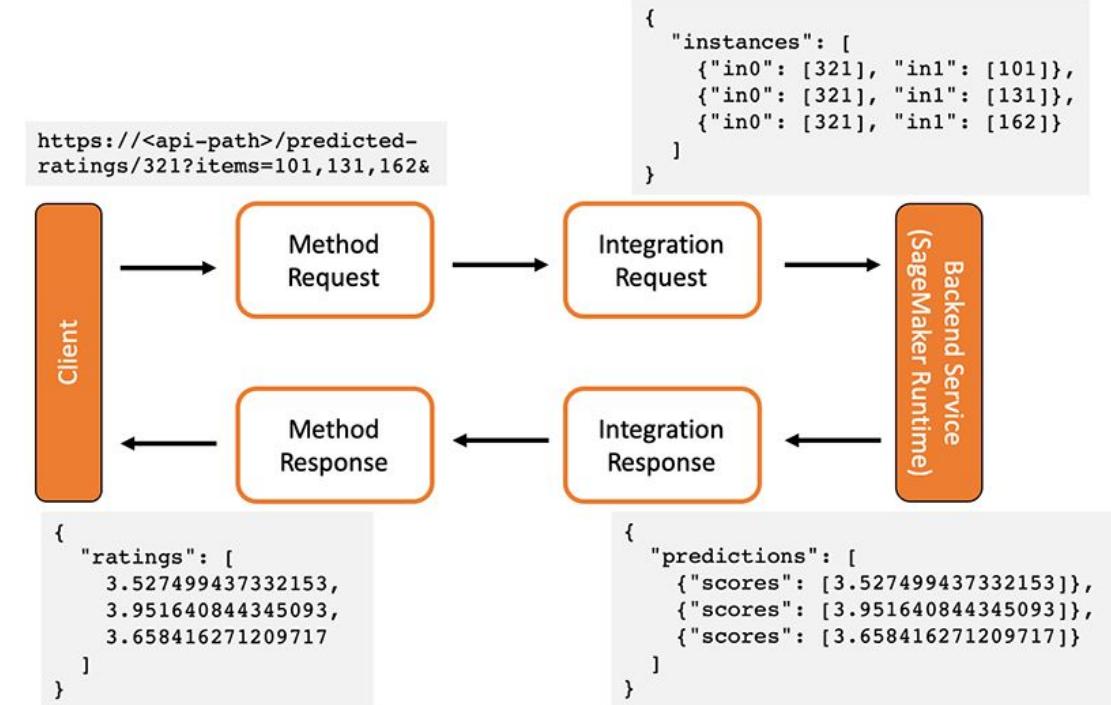
- **Son procesos que estaban siempre activos en espera y que se pueden ejecutar siempre mediante una petición**
 - P.e. se puede realizar una predicción cada vez que llegue un dato sin necesidad de esperar a la siguiente ejecución (si fuese en batch).
- **Se suele enviar muestras de una en una para hacer las predicciones**
 - Se suele enviar una muestra para realizar la predicción de la misma.



Real-Time

Generalmente para usar un modelo, lo que se hace es crear una API Rest que permita hacer predicciones con el mismo.

```
url = 'http://127.0.0.1:5000/'  
params ={'query': 'that movie was boring'}  
response = requests.get(url, params)  
response.json()  
  
Output: {'confidence': 0.128, 'prediction': 'Negative'}
```



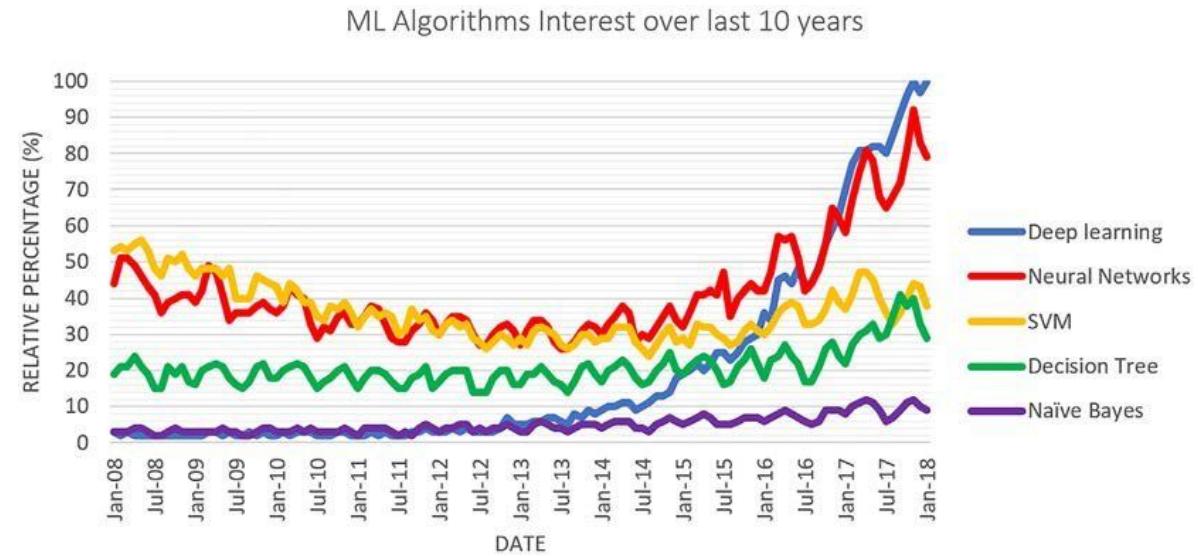
Índice de la sesión

- Recap
- **Introducción a las redes neuronales**
- Configuración de los modelos de redes neuronales
- Modelos de Deep Learning



Introducción a las redes neuronales

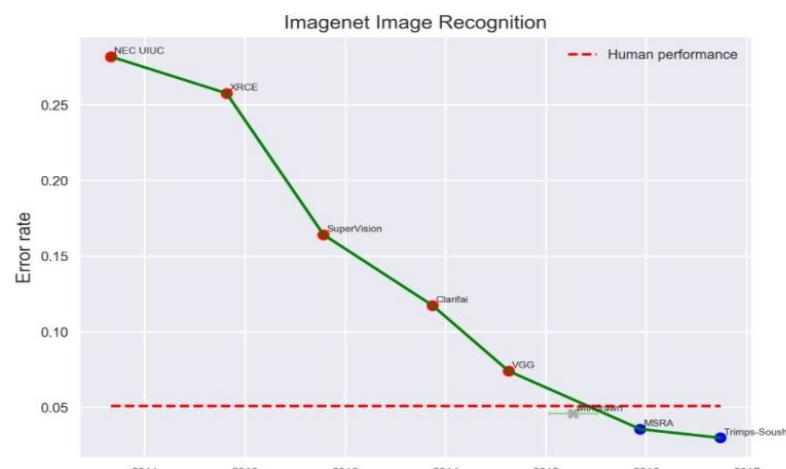
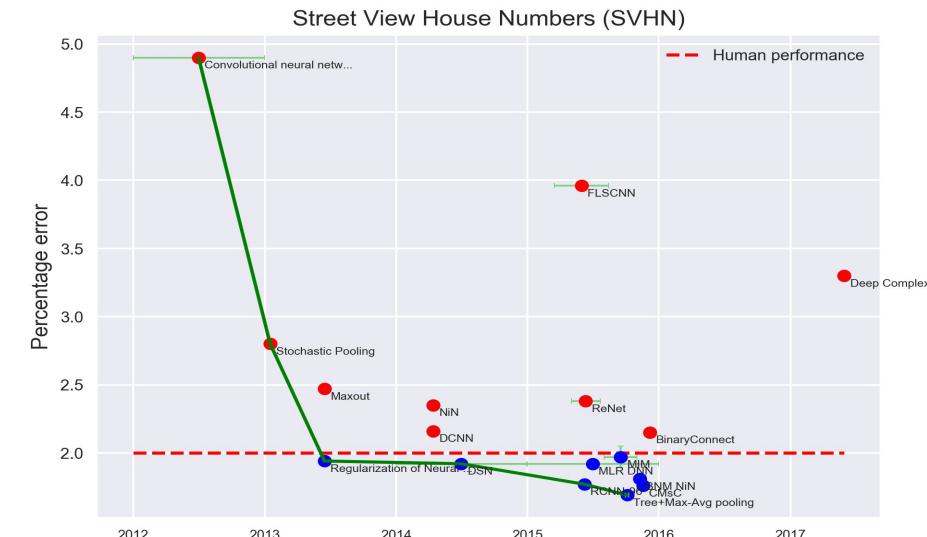
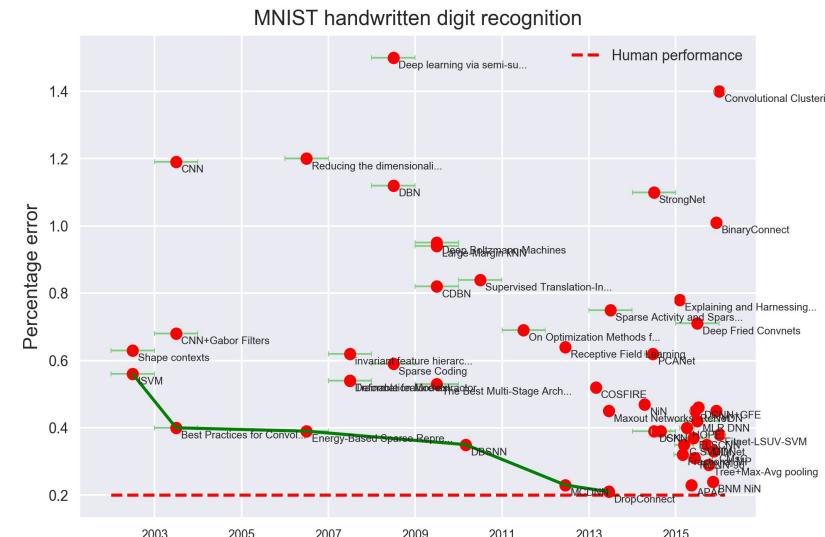
Los modelos de Deep Learning han copado gran parte de las búsquedas en los proyectos con Machine Learning.



[source](#)

Introducción a las redes neuronales

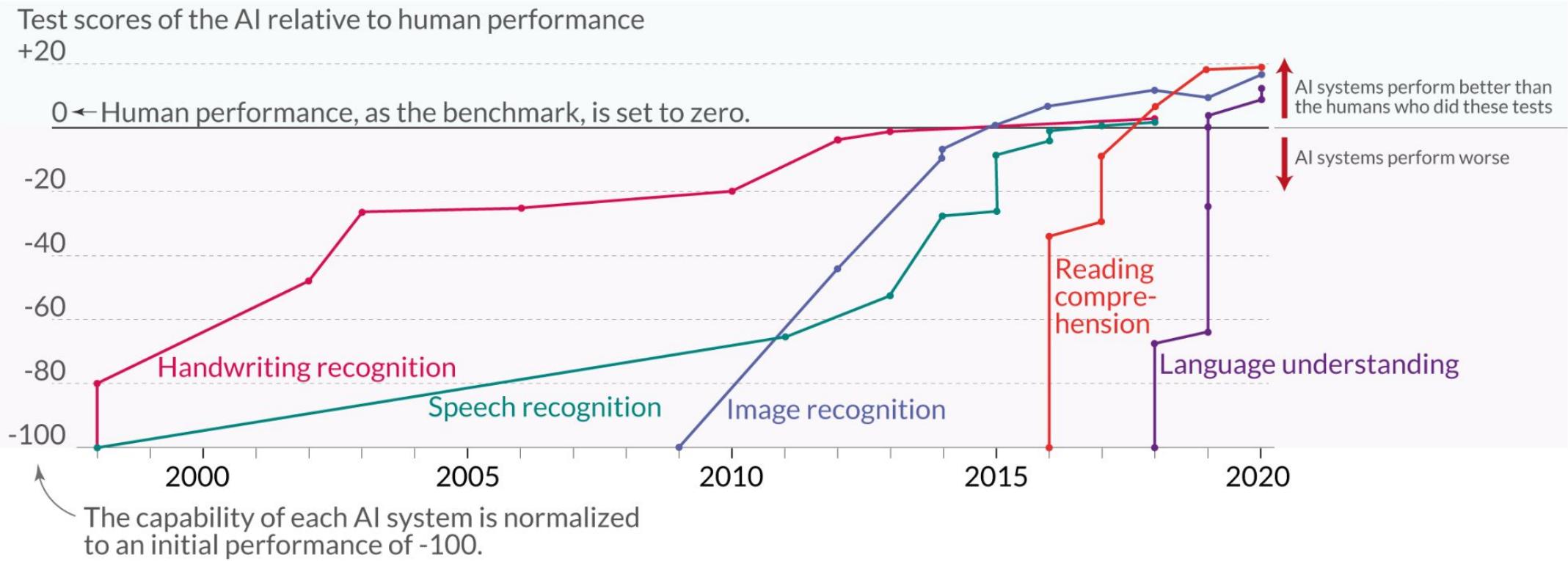
El performance medio de determinados modelos ha superado al de los humanos en algunas tareas.



En los últimos años hemos dado un salto exponencial

Language and image recognition capabilities of AI systems have improved rapidly

Our World
in Data



Data source: Kiela et al. (2021) – Dynabench: Rethinking Benchmarking in NLP
OurWorldinData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the author Max Roser

Esta revolución va a cambiar nuestra manera de trabajar (y los trabajos)

Group	Occupations with highest exposure	% Exposure
Human α	Interpreters and Translators	76.5
	Survey Researchers	75.0
	Poets, Lyricists and Creative Writers	68.8
	Animal Scientists	66.7
	Public Relations Specialists	66.7
Human β	Survey Researchers	84.4
	Writers and Authors	82.5
	Interpreters and Translators	82.4
	Public Relations Specialists	80.6
	Animal Scientists	77.8
Human ζ	Mathematicians	100.0
	Tax Preparers	100.0
	Financial Quantitative Analysts	100.0
	Writers and Authors	100.0
	Web and Digital Interface Designers	100.0
	<i>Humans labeled 15 occupations as "fully exposed."</i>	
Model α	Mathematicians	100.0
	Correspondence Clerks	95.2
	Blockchain Engineers	94.1
	Court Reporters and Simultaneous Captioners	92.9
	Proofreaders and Copy Markers	90.9
Model β	Mathematicians	100.0
	Blockchain Engineers	97.1
	Court Reporters and Simultaneous Captioners	96.4
	Proofreaders and Copy Markers	95.5
	Correspondence Clerks	95.2
Model ζ	Accountants and Auditors	100.0
	News Analysts, Reporters, and Journalists	100.0
	Legal Secretaries and Administrative Assistants	100.0
	Clinical Data Managers	100.0
	Climate Change Policy Analysts	100.0
	<i>The model labeled 86 occupations as "fully exposed."</i>	
Highest variance	Search Marketing Strategists	14.5
	Graphic Designers	13.4
	Investment Fund Managers	13.0
	Financial Managers	13.0
	Insurance Appraisers, Auto Damage	12.6

Job Zone	Preparation Required	Education Required	Example Occupations	Median Income	Tot Emp (000)	H α		M β		H ζ	
						M	α	H	M	β	ζ
1	None or little (0-3 months)	High school diploma or GED (optional)	Food preparation workers, dishwashers, floor sanders	\$30,230	13,100	3.71	3.84	6.45	5.97	9.19	8.11
2	Some (3-12 months)	High school diploma	Orderlies, customer service representatives, tellers	\$38,215	73,962	7.03	11.88	15.74	19.54	24.45	27.19
3	Medium (1-2 years)	Vocational school, on-the-job training, or associate's degree	Electricians, barbers, medical assistants	\$54,815	37,881	11.28	13.72	26.08	32.17	40.88	50.62
4	Considerable (2-4 years)	Bachelor's degree	Database administrators, graphic designers, cost estimators	\$77,345	56,833	22.68	17.82	46.78	51.30	70.87	84.78
5	Extensive (4+ years)	Master's degree or higher	Pharmacists, lawyers, astronomers	\$81,980	21,221	22.81	13.36	43.11	44.64	63.41	75.92

Table 6: Exposure to GPTs by Job Zone

GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models
Tyna Eloundou et al.

Introducción a las redes neuronales

Nuevas herramientas como chatGPT, GPT-3, GPT-4, Dalle, etc... son modelos de NLP creados a partir de la arquitectura de transformers que son un tipo de modelos basados en Deep Learning.



3D render of a cute tropical fish in an aquarium on a dark blue background, digital art

podrías hacer una canción que fuese un mix entre 50cent y la sirenita?

(Verse 1 - estilo 50 Cent)
Yo, bajo el mar es donde me encuentro,
Nadando con peces, mi nuevo talento,
Ariel, la sirenita, es mi amiga,
Juntos en el agua, la vida sigue arriba.

Pre-estribillo:
Uh huh, sé que suena loco,
Pero en este mundo mágico, todo es poco,
Sirenas y raperos, desafiando el tiempo,
Al ritmo de las olas, creando este momento.

(Estribillo - estilo La Sirenita)
Bajo el mar, bajo el mar,
Donde todo es mejor y nadie puede negar,
En la superficie, sufren, no pueden respirar,
Junto a 50 Cent, nadamos sin parar.

(Verse 2 - estilo 50 Cent)
Soy el rey del rap en esta vida submarina,
Con Ariel a mi lado, brillamos como espina,
Tiburones y pulpos, no pueden competir,

Transformers

Paper en 2017: Attention is all you need, que introdujo la arquitectura de los transformers.

The screenshot shows the arXiv website interface for the paper "Attention Is All You Need". The page title is "arXiv > cs > arXiv:1706.03762". The main content area displays the paper's abstract, authors (Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin), and a summary of its performance. Below the abstract are sections for comments, subjects, and citation details. The right sidebar provides download options (PDF, Other formats), a current browse context (cs.CL), blog links, references, and citation information. A large green circular graphic is overlaid on the right side of the screenshot.

Computer Science > Computation and Language
[Submitted on 12 Jun 2017 (v1), last revised 6 Dec 2017 (this version, v5)]

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Comments: 15 pages, 5 figures
Subjects: Computation and Language (cs.CL); Machine Learning (cs.LG)
Cite as: arXiv:1706.03762 [cs.CL]
(or arXiv:1706.03762v5 [cs.CL] for this version)
<https://doi.org/10.48550/arXiv.1706.03762>

Submission history
From: Ashish Vaswani [view email]
[v1] Mon, 12 Jun 2017 17:57:34 UTC (1,102 KB)
[v2] Mon, 19 Jun 2017 16:49:45 UTC (1,125 KB)
[v3] Tue, 20 Jun 2017 05:20:02 UTC (1,125 KB)
[v4] Fri, 30 Jun 2017 17:29:30 UTC (1,124 KB)
[v5] Wed, 6 Dec 2017 03:30:32 UTC (1,124 KB)

Download:

- PDF
- Other formats

(license)

Current browse context:
cs.CL
< prev | next >
new | recent | 1706
Change to browse by:
cs
cs.LG

References & Citations

- NASA ADS
- Google Scholar
- Semantic Scholar

66 blog links (what is this?)

DBLP - CS Bibliography
listing | bibtex
Ashish Vaswani
Noam Shazeer
Niki Parmar
Jakob Uszkoreit
Llion Jones
...
Export BibTeX Citation

Bookmark

La idea de los transformers está basada en el concepto de self-attention, que se refiere a una serie de mecanismos diseñados para "atender" partes de la oración en el contexto de otras oraciones, generalmente para la realización de tareas de predicción.

Transformers

Los transformers buscan encontrar aquellas partes del texto más importantes en el contexto, que proporcionan mayor información. Ejemplo con Visual Attention.

Neural Image Caption Generation with Visual Attention

Figure 4. Examples of attending to the correct object (white indicates the attended regions, *underlines* indicated the corresponding word)



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Transformers

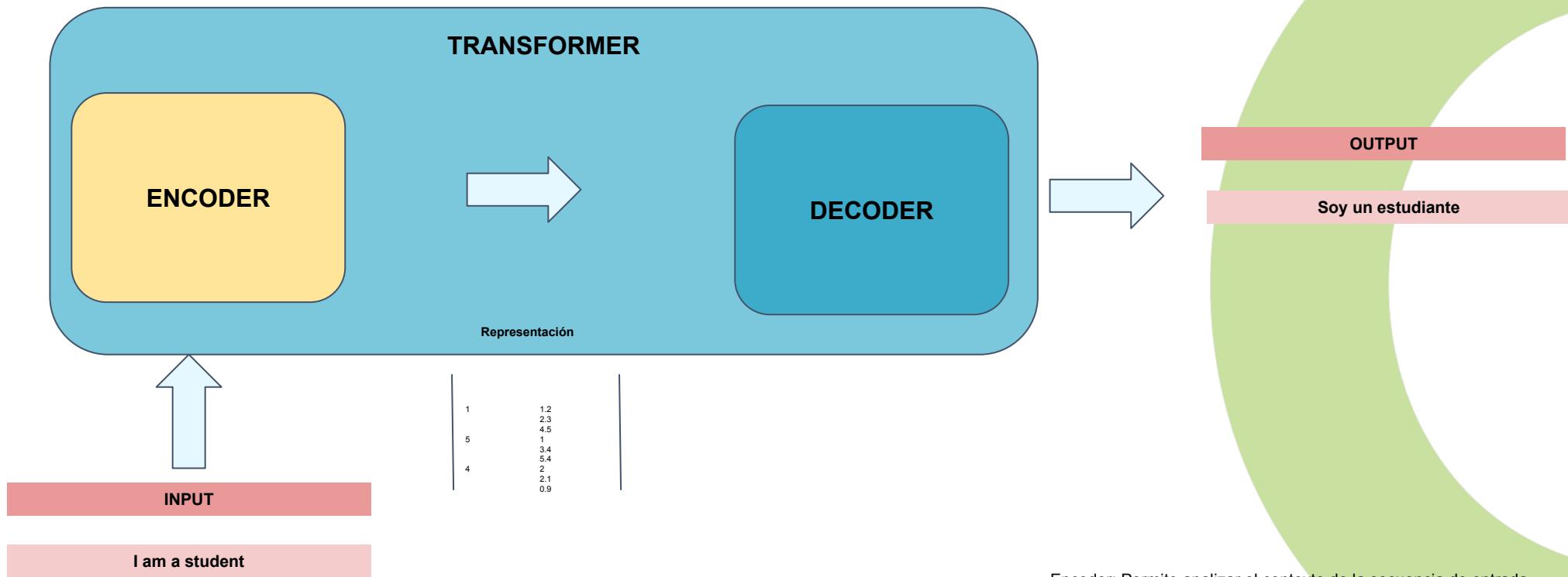
Pensemos por ejemplo en el contexto de esta frase y en la palabra "Él".

Mi colega y yo nos fuimos de ruta a la montaña. Él acabó cansado.

¿A quién se refiere él? ¿Es realmente un modelo convencional capaz de captar esa información?

Transformers

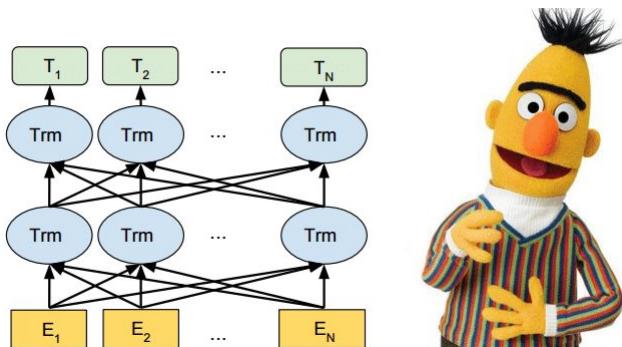
Los transformers consisten en un encoder y un decoder. El input pasa por el encoder y genera una representación matricial de este. El decoder coge esa representación y genera un output.



BERT

Ejemplo práctico de uso con pipelines de BERT.

<https://colab.research.google.com/drive/1IC9QoFJSKZQp7dXA60Xmw6wN0Ve7EqVt?usp=sharing>



BERT

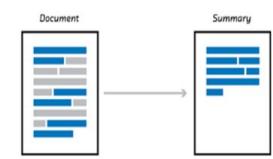
Ejercicio práctico.

**haz un resumen del texto con BERT de la primera sección de:
https://en.wikipedia.org/wiki/History_of_Spain**

Aplicación de los Transformers

Existen múltiples aplicaciones de los transformers. En algunos casos se utiliza el encoder y decoder por separado para diferentes fines.

- Machine Learning Translation
- Document summarization
- Name Entity Recognition
- Video Understanding
- Analysis
- Biological Sequences

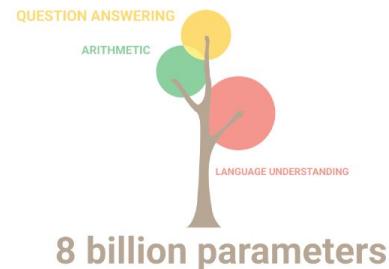


In fact, the Chinese **tech** market has the **three** **CARDINAL** most influential names of the retail and tech space - **Alibaba** **ORG**, **Baidu** **ORG**, and **Tencent** **PERSON** (collectively touted as **BAT** **CO**), and is betting big in the global **AI** **ORG** in retail industry space. The **three** **CARDINAL** giants which are claimed to have a cut-throat competition with the **US** **ORG** (in terms of resources and capital) are positioning themselves to become the future **AI** **PERSON** platforms. The trio is also expanding in other **Asian** **LOC** countries and investing heavily in the **US** **ORG** based **AI** **ORG** startups to leverage the power of **AI** **ORG**. Backed by such powerful initiatives and presence of these conglomerates, the market in APAC AI is forecast to be the fastest-growing **one** **CARDINAL**, with an anticipated **CAGR** **PERIOD** of **45%** **PERCENT** over **2019-2024** **DATES**.

To further elaborate on the geographical trends, North America **LOC** has procured **more than 50%** **PERCENT** of the global share in **2017** **DATES** and has been leading the regional landscape of **AI** **ORG** in the retail market. The **US** **ORG** has a significant credit in the regional trends with **over 65%** **PERCENT** of investments (including M&As, private equity, and venture capital) in artificial intelligence technology. Additionally, the region is a huge hub for startups in tandem with the presence of tech titans, such as **Google** **ORG**, **IBM** **ORG**, and **Microsoft** **ORG**.

Crecimiento de los modelos de LLM

Estamos viviendo un momento de competitividad entre las Big Tech, en los que cada vez vemos modelos más y más grandes.



Estos modelos necesitan mucho tiempo de entrenamiento.



Crecimiento de los modelos de LLM

Hacer esta tipología de modelos tiene unos tiempos de cómputo elevados.

D Total Compute Used to Train Language Models

This appendix contains the calculations that were used to derive the approximate compute used to train the language models in Figure 2.2. As a simplifying assumption, we ignore the attention operation, as it typically uses less than 10% of the total compute for the models we are analyzing.

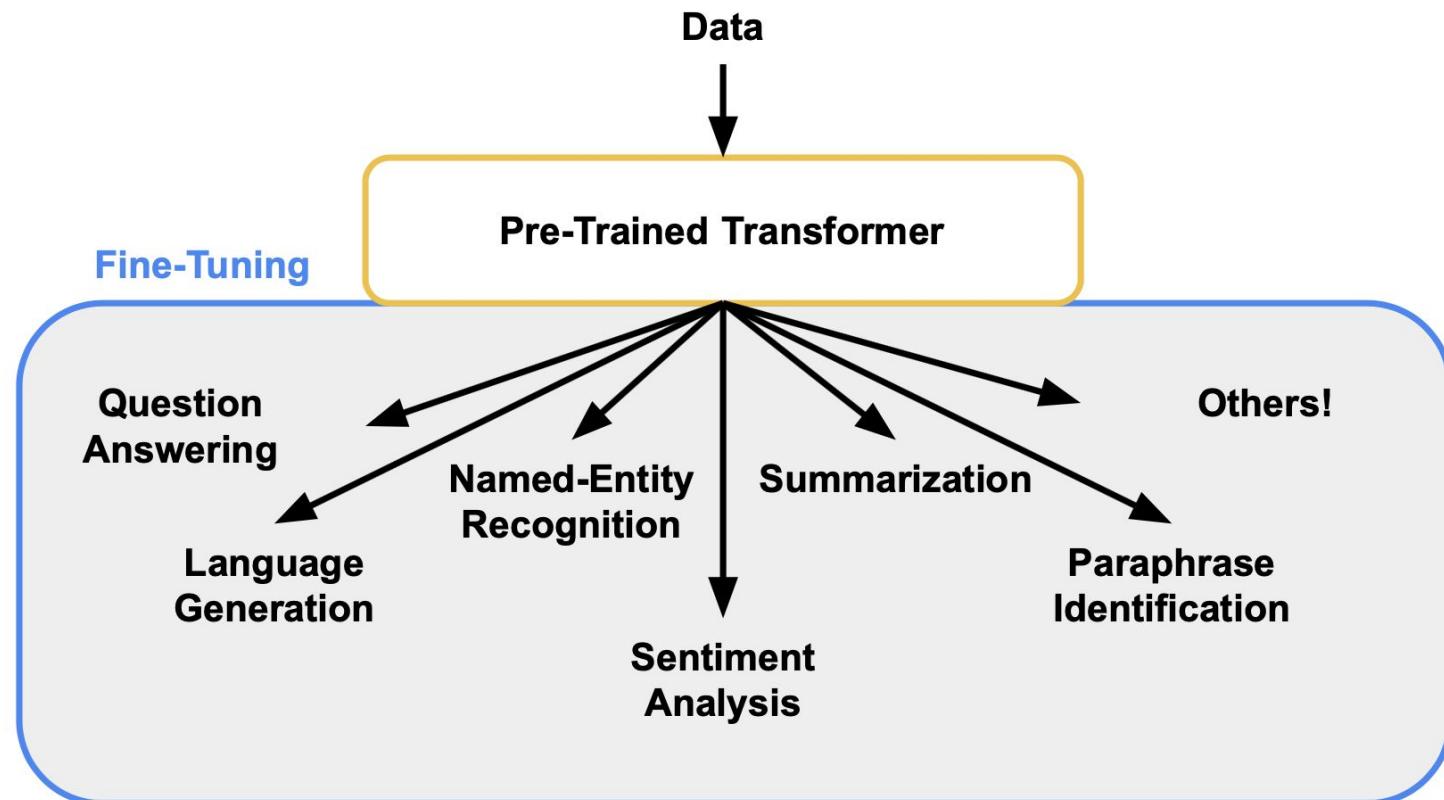
Calculations can be seen in Table D.1 and are explained within the table caption.

Model	Total train compute (PF-days)	Total train compute (flops)	Params (M)	Training tokens (billions)	Flops per param per token	Mult for bwd pass	Fwd-pass flops per active param per token	Frac of params active for each token
T5-Small	2.08E+00	1.80E+20	60	1,000	3	3	1	0.5
T5-Base	7.64E+00	6.60E+20	220	1,000	3	3	1	0.5
T5-Large	2.67E+01	2.31E+21	770	1,000	3	3	1	0.5
T5-3B	1.04E+02	9.00E+21	3,000	1,000	3	3	1	0.5
T5-11B	3.82E+02	3.30E+22	11,000	1,000	3	3	1	0.5
BERT-Base	1.89E+00	1.64E+20	109	250	6	3	2	1.0
BERT-Large	6.16E+00	5.33E+20	355	250	6	3	2	1.0
RoBERTa-Base	1.74E+01	1.50E+21	125	2,000	6	3	2	1.0
RoBERTa-Large	4.93E+01	4.26E+21	355	2,000	6	3	2	1.0
GPT-3 Small	2.60E+00	2.25E+20	125	300	6	3	2	1.0
GPT-3 Medium	7.42E+00	6.41E+20	356	300	6	3	2	1.0
GPT-3 Large	1.58E+01	1.37E+21	760	300	6	3	2	1.0
GPT-3 XL	2.75E+01	2.38E+21	1,320	300	6	3	2	1.0
GPT-3 2.7B	5.52E+01	4.77E+21	2,650	300	6	3	2	1.0
GPT-3 6.7B	1.39E+02	1.20E+22	6,660	300	6	3	2	1.0
GPT-3 13B	2.68E+02	2.31E+22	12,850	300	6	3	2	1.0
GPT-3 175B	3.64E+03	3.14E+23	174,600	300	6	3	2	1.0

Estimación considerando una NVIDIA Tesla V100 GPU.

Crecimiento de los modelos de LLM

Por ello, cada vez más, el uso de modelos de NLP se reduce al uso de una API o a hacer un fine-tuning de los modelos.

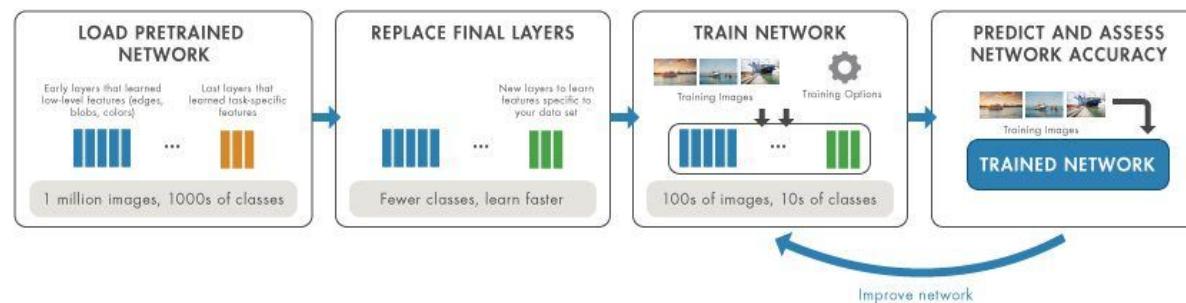


Realización de fine tuning de un modelo

¿Qué significa hacer un modelo de transfer learning?

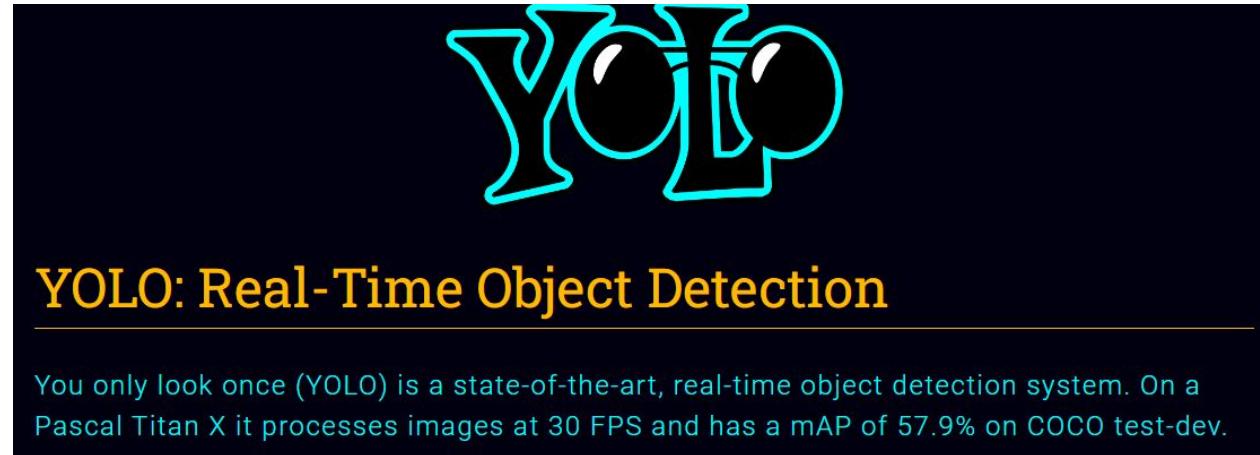
Cuando tenemos un modelo pre-entrenado, podemos realizar las siguientes técnicas para adecuar el modelo genérico a un objetivo determinado:

1. Actualizar todo el modelo sobre los datos etiquetados + cualquier capa adicional añadida encima.
2. Actualizar un subconjunto de capas para ahorrar tiempo de cálculo.
3. Congelar todo el modelo de capas y sólo entrenar las capas adicionales añadidas encima.



Modelos de Computer Vision

Sucede algo similar con los modelos de Computer Vision. La tendencia cada vez más radica en el uso de modelos pre-entrenados, en los que con pocos datos, es posible conseguir un buen resultado.



Ejemplo de API de Google Computer Vision

Ejemplo en Computer Vision.

Ejemplo con imágenes

1. Seleccionamos una imagen:
 - [Ejemplo 1](#)
 - [Ejemplo 2](#)
 - [Ejemplo 3](#)
2. Entramos a la página web [Google Images Drag & Drop](#).
3. Introducimos la foto en la web.

Pregunta

¿Qué piensas que puede recomendar un algoritmo de redes sociales que sabe esta información extraída de las fotos?

Ejemplo de API con NLP

Ejemplo en NLP.

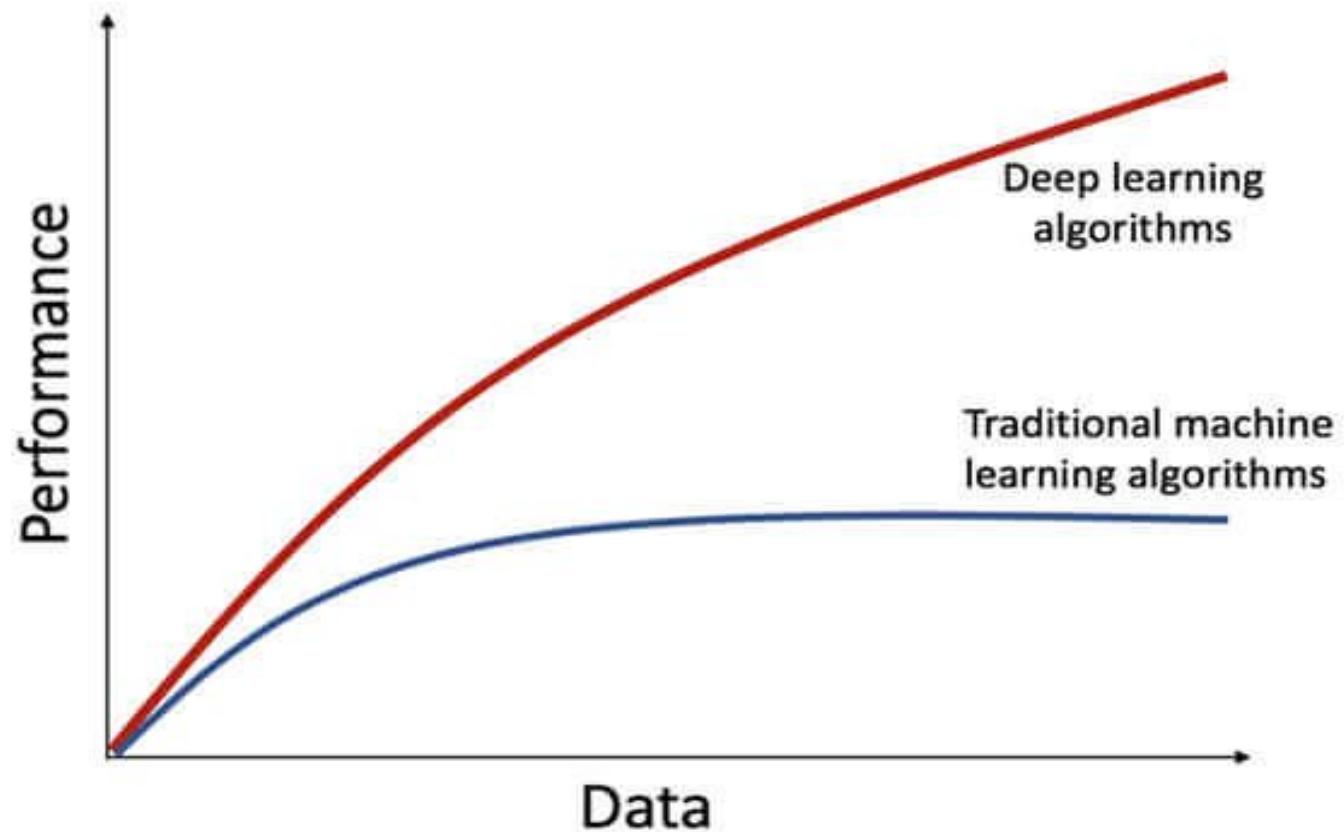
Veamos un ejemplo del valor que se puede obtener de una imagen.

Ejemplo con texto

1. Texto:
 - Don Quijote de la Mancha es una novela escrita por el español Miguel de Cervantes Saavedra. Publicada su primera parte con el título de El ingenioso hidalgo don Quijote de la Mancha a comienzos de 1605, es la obra más destacada de la literatura española y una de las principales de la literatura universal, además de ser la más leída después de la Biblia
2. Entramos a la página web [Google NLP](#).
3. Introducimos el texto.

Introducción a las redes neuronales

Performance en datos.



Frameworks de Deep Learning

Popularidad de las Librerías para el uso de Deep Learning.

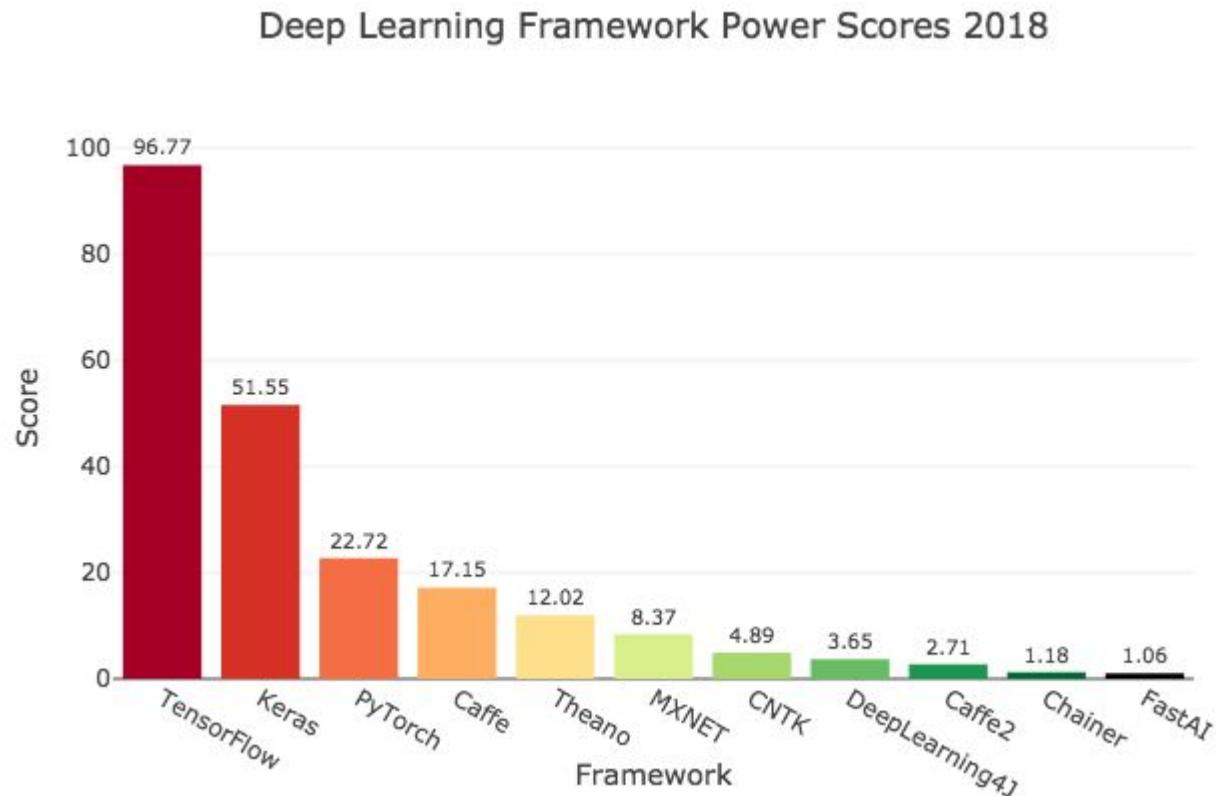


TensorFlow



Frameworks de Deep Learning

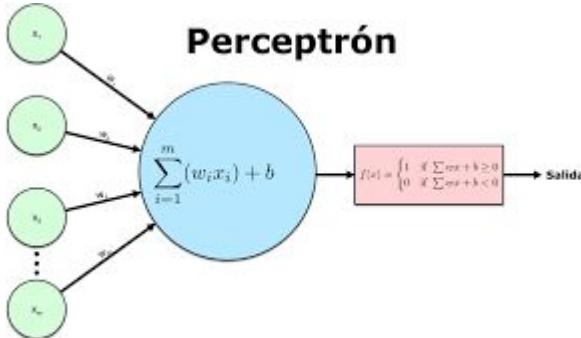
Popularidad de las Librerías para el uso de Deep Learning en 2018.



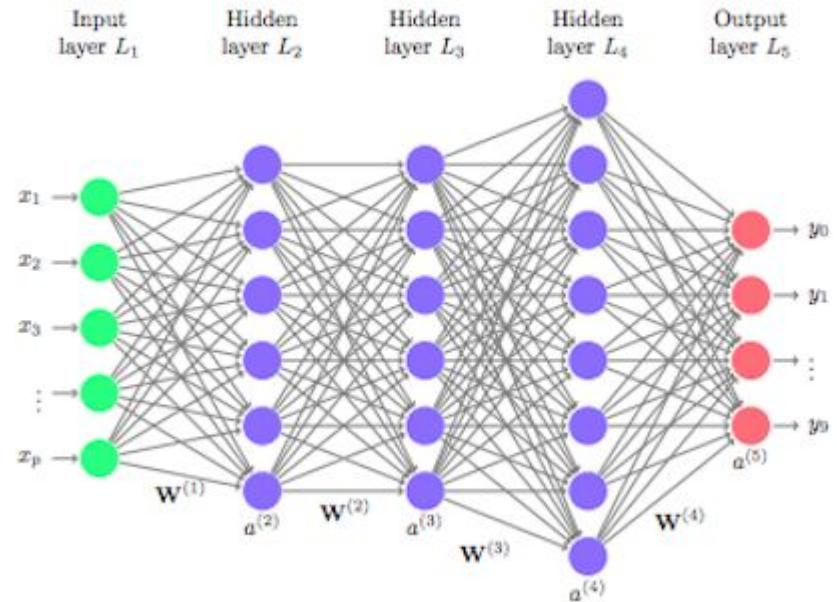
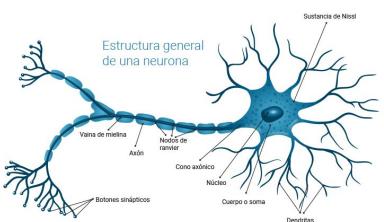
Usaremos keras en esta
sesión

Deep Learning

Las neuronas (o perceptrones) son las unidades básicas en los modelos de redes neuronales.



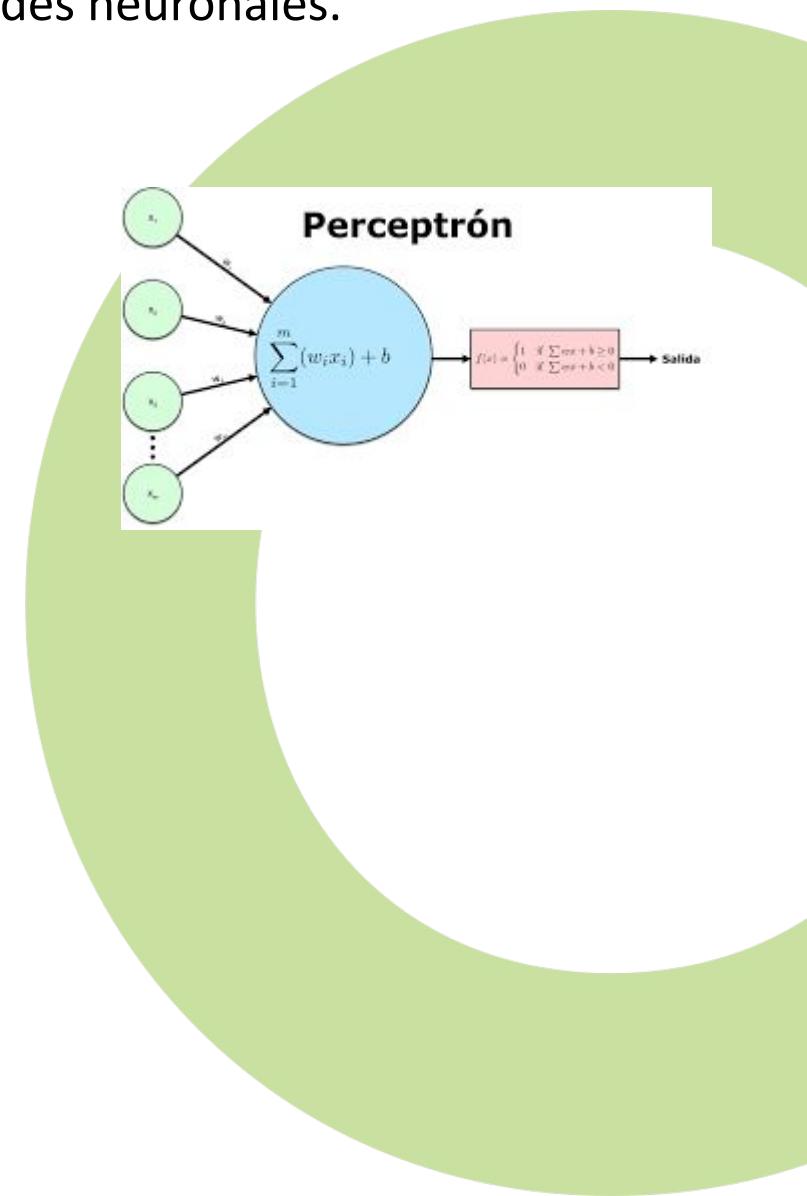
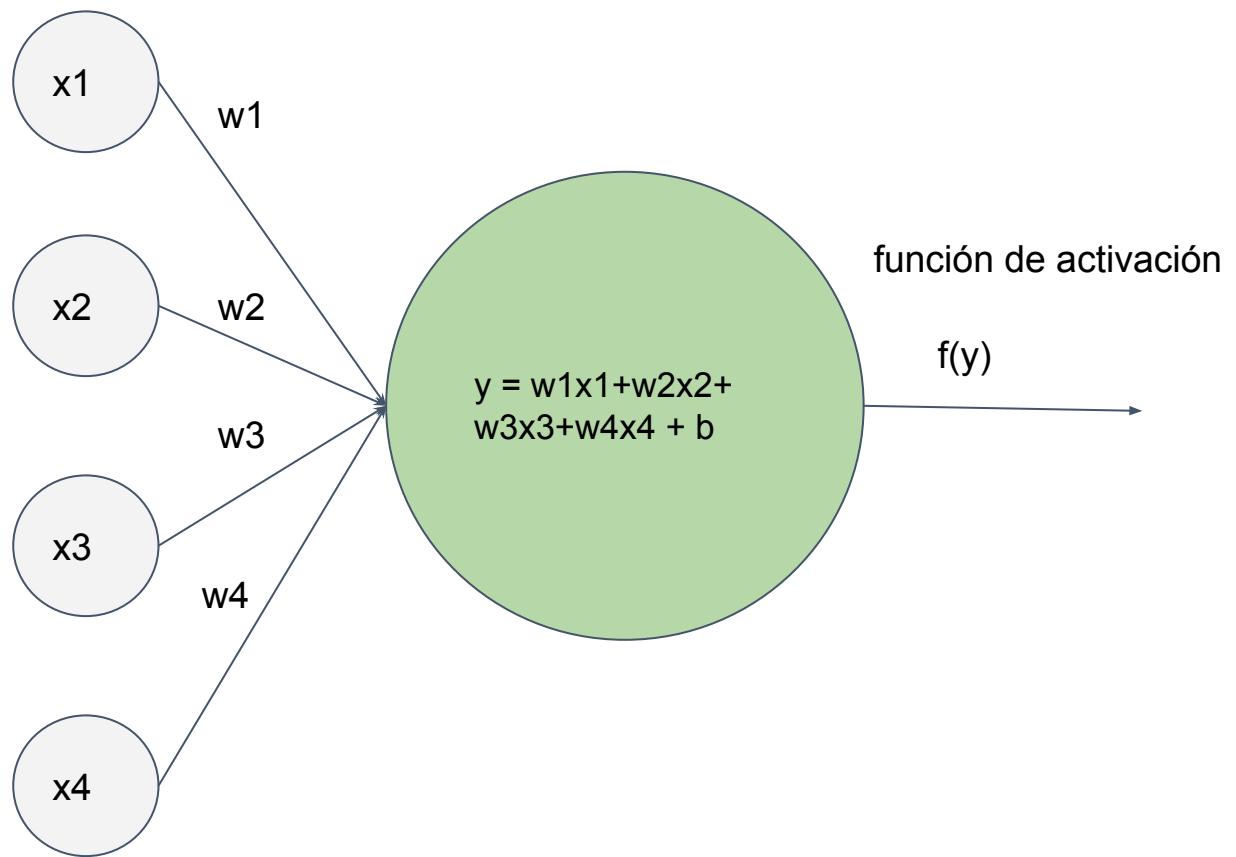
La unidad fundamental de las redes neuronales es el perceptrón.



Las redes neuronales son un conjunto de funciones no lineales dependientes. Cada función individual consiste en una neurona (o un perceptrón). En las capas totalmente conectadas, la neurona aplica una transformación lineal al vector de entrada a través de una matriz de pesos. A continuación, se aplica una transformación no lineal al producto a través de una función de activación no lineal f.

Deep Learning

Las neuronas (o perceptrones) son las unidades básicas en los modelos de redes neuronales.



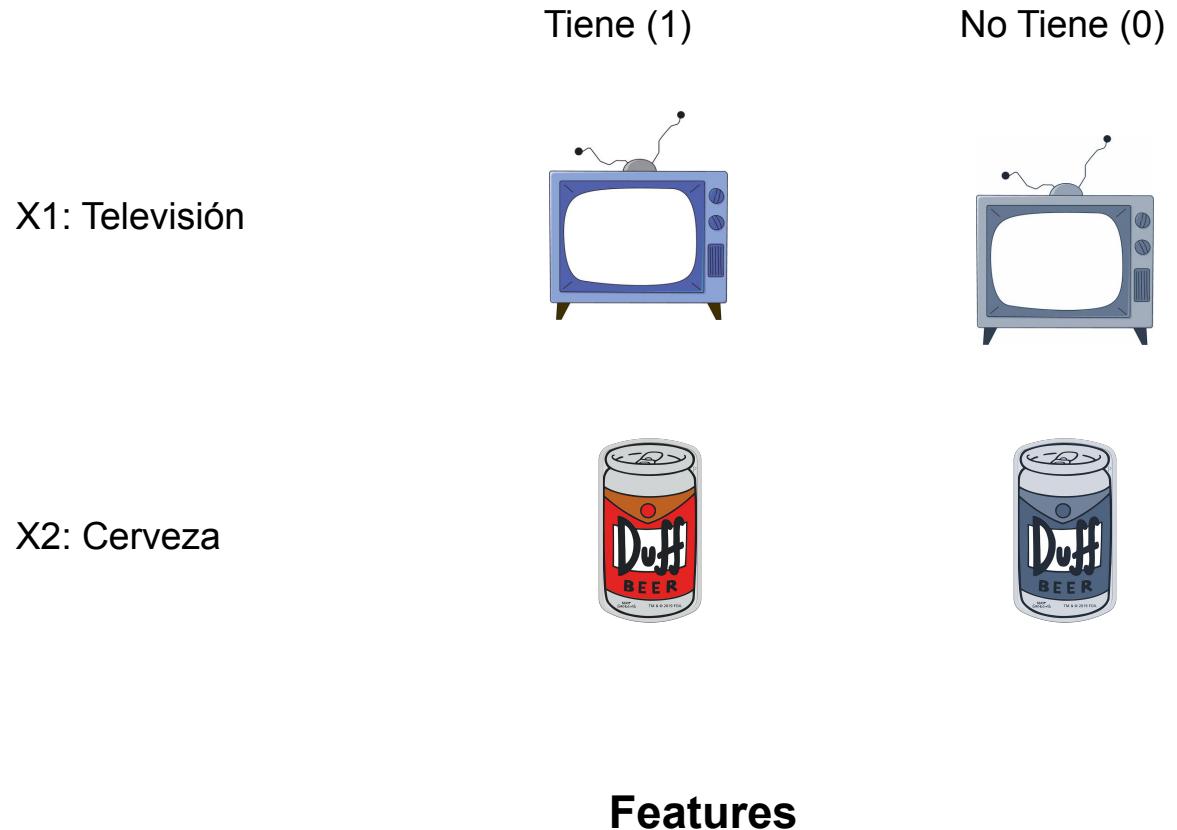
Deep Learning

Ejemplo práctico para entender una neurona: Vamos a predecir si Homer pierde la cabeza con una neurona.



Deep Learning

Ejemplo práctico: Las variables que tenemos son X1 (si tiene o no televisión) y X2 (si tiene o no cerveza).



Deep Learning

Ejemplo práctico: El target es Y (si pierde o no pierde la cabeza).

Y: Cabeza

La pierde (1)



No la pierde (0)

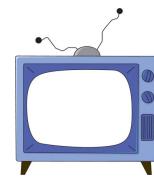


target

Deep Learning

Ejemplo práctico: Buscamos la combinación tal que X1 sea no tiene televisión y X2 no tiene cerveza-> pierde la cabeza.

X1: Televisión X2: Cerveza target



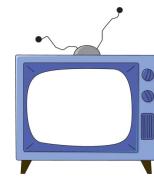
1



1



0



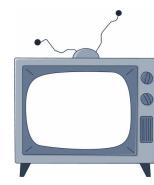
1



0



0



0



1



0



0



0

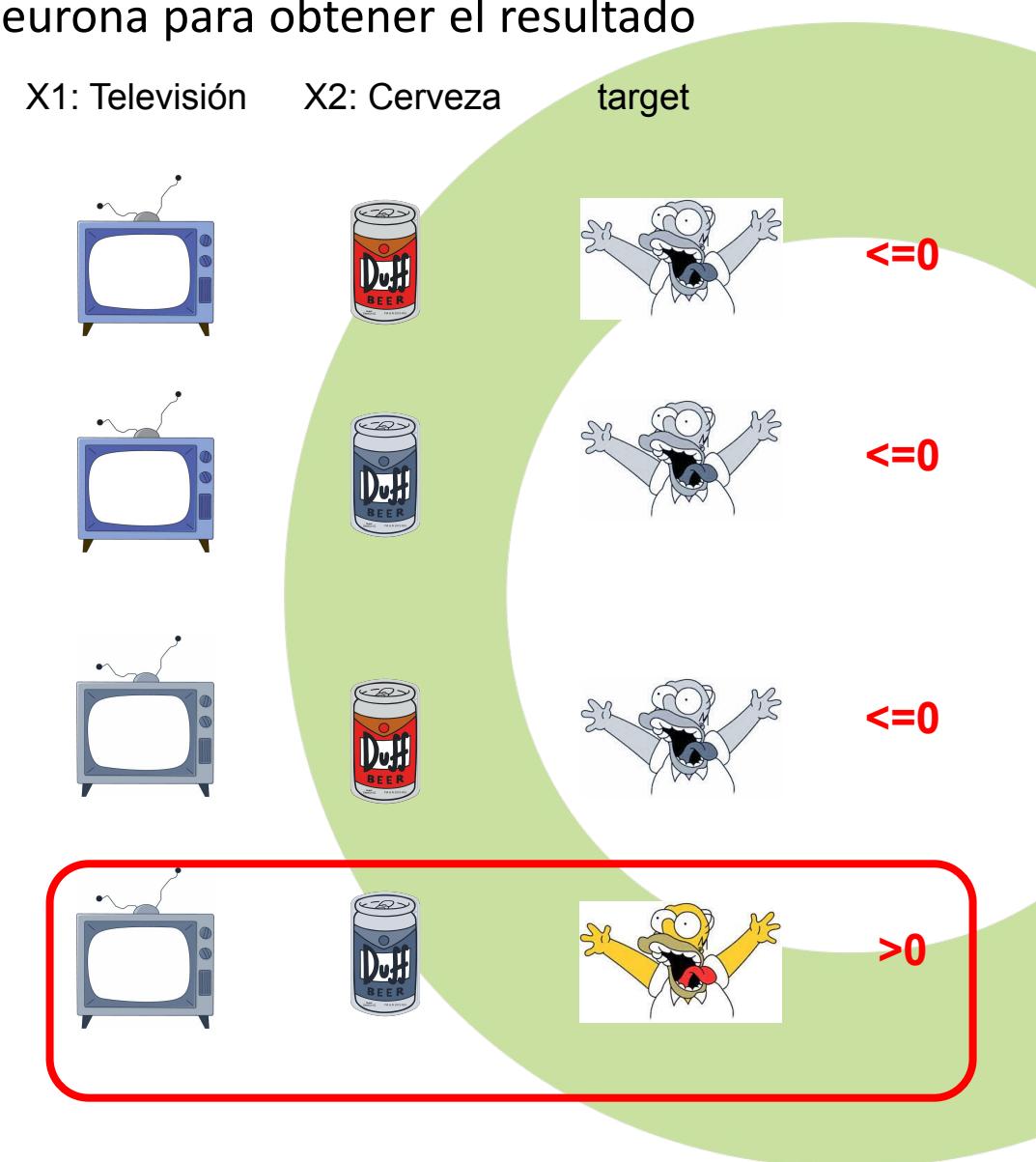
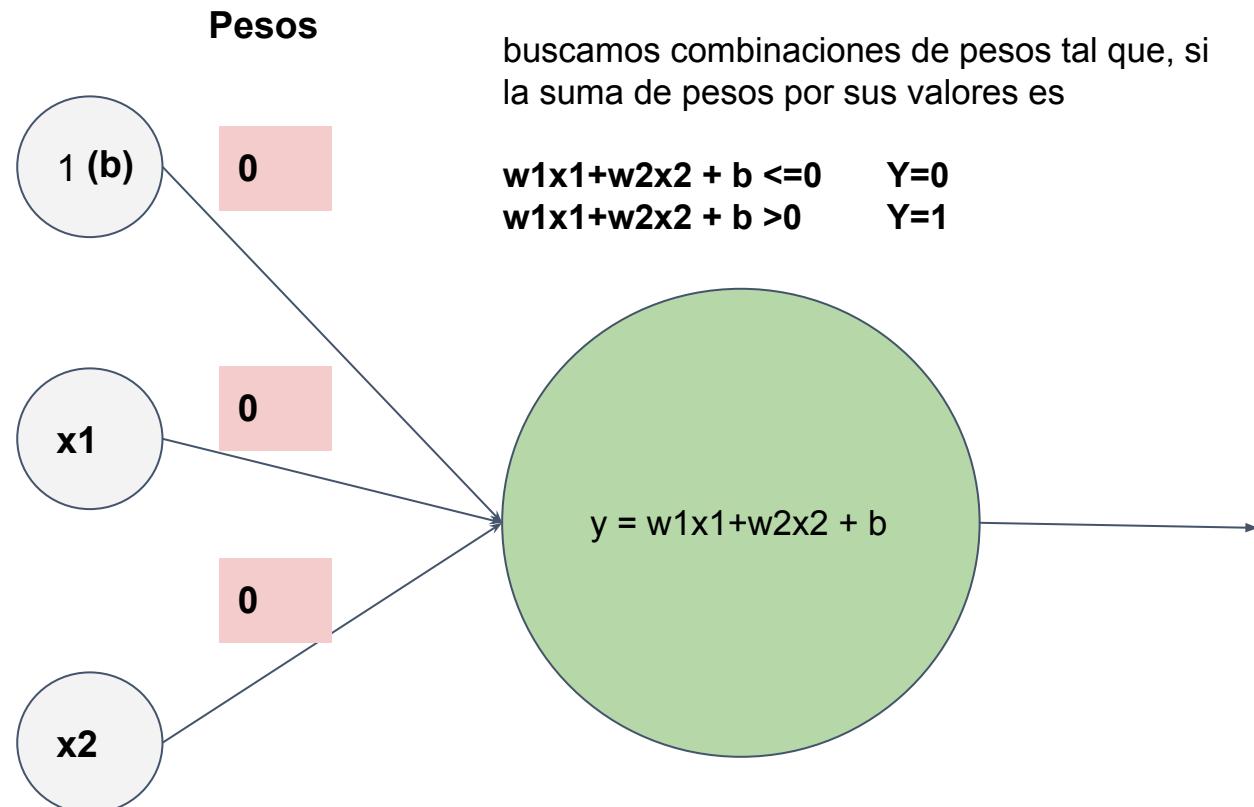


1

Vamos a crear un modelo de 1 neurona que sea capaz de predecir en función de X1 y X2, si Homer pierde la cabeza.

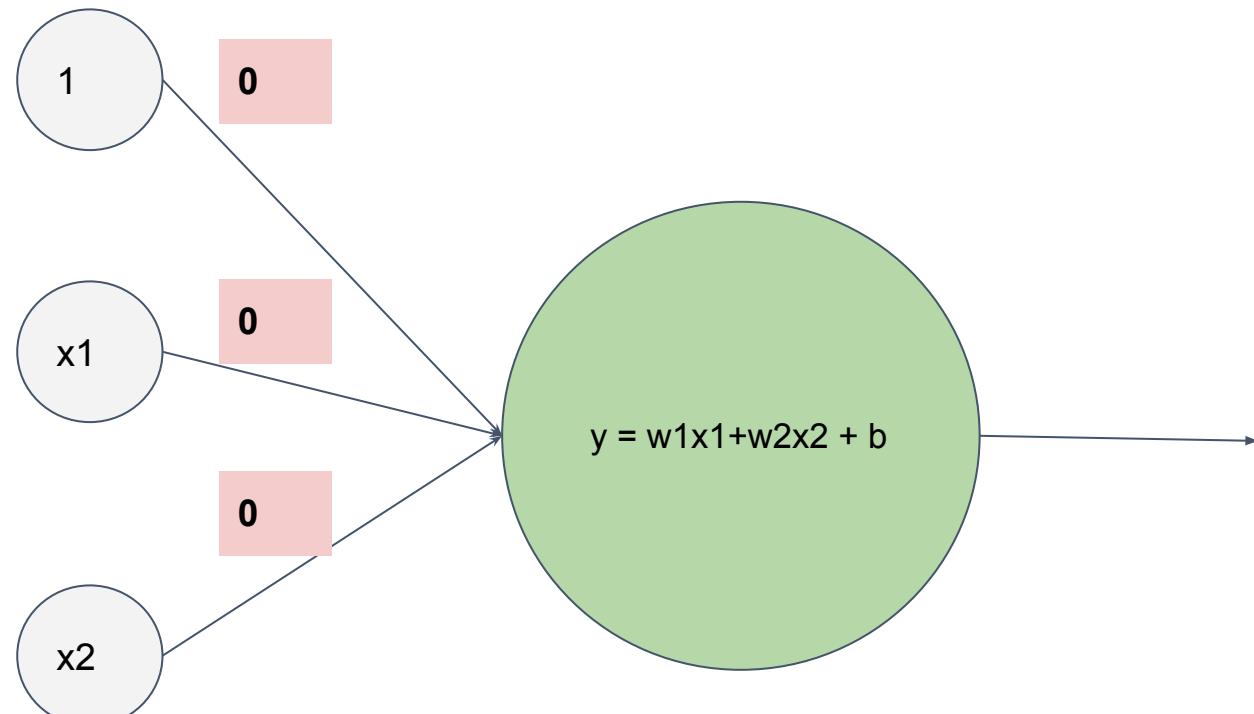
Deep Learning

Ejemplo práctico: Para ello, vamos a **modificar los pesos** de la neurona para obtener el resultado de la derecha y ver si se ajusta con nuestro modelo.



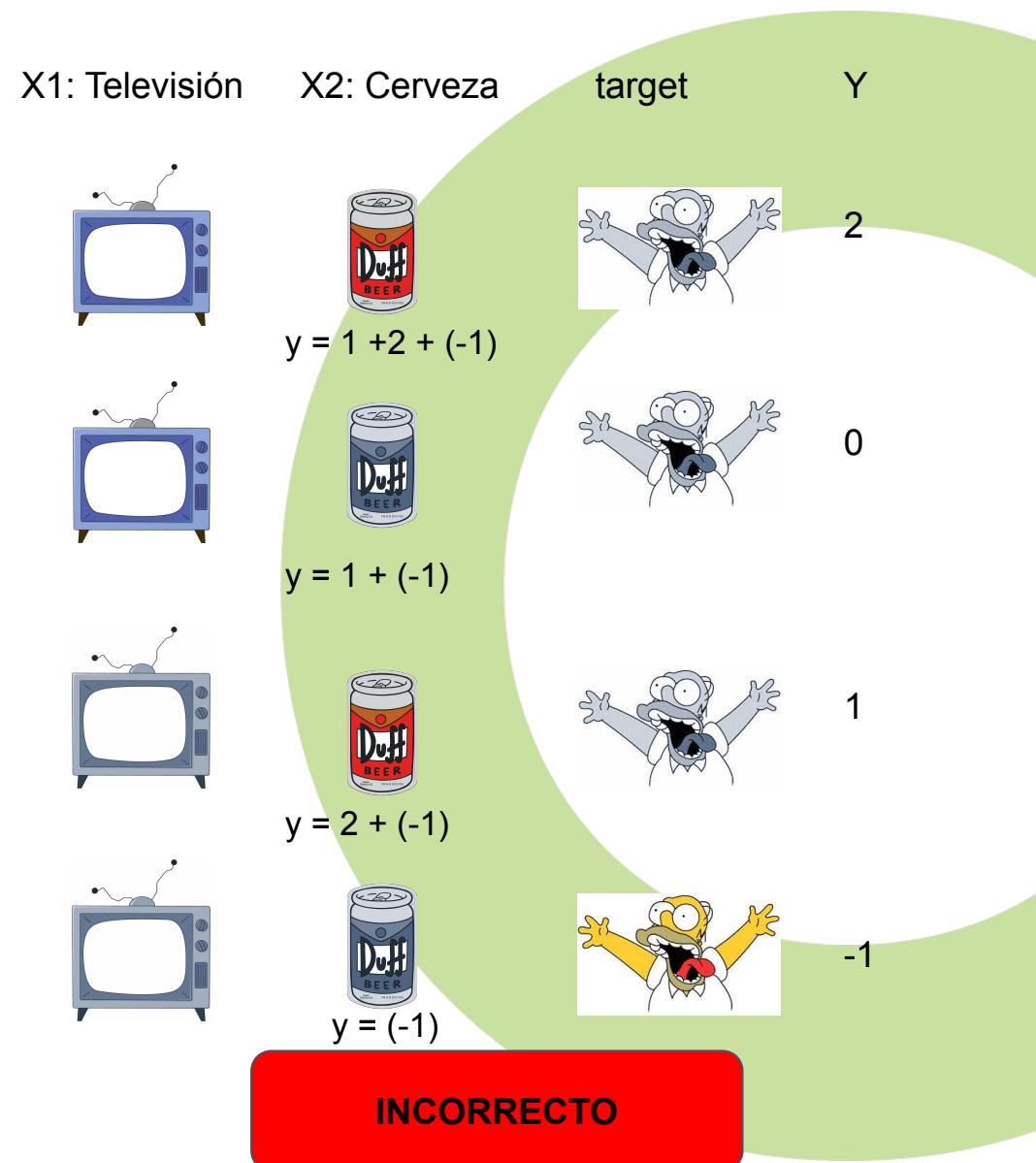
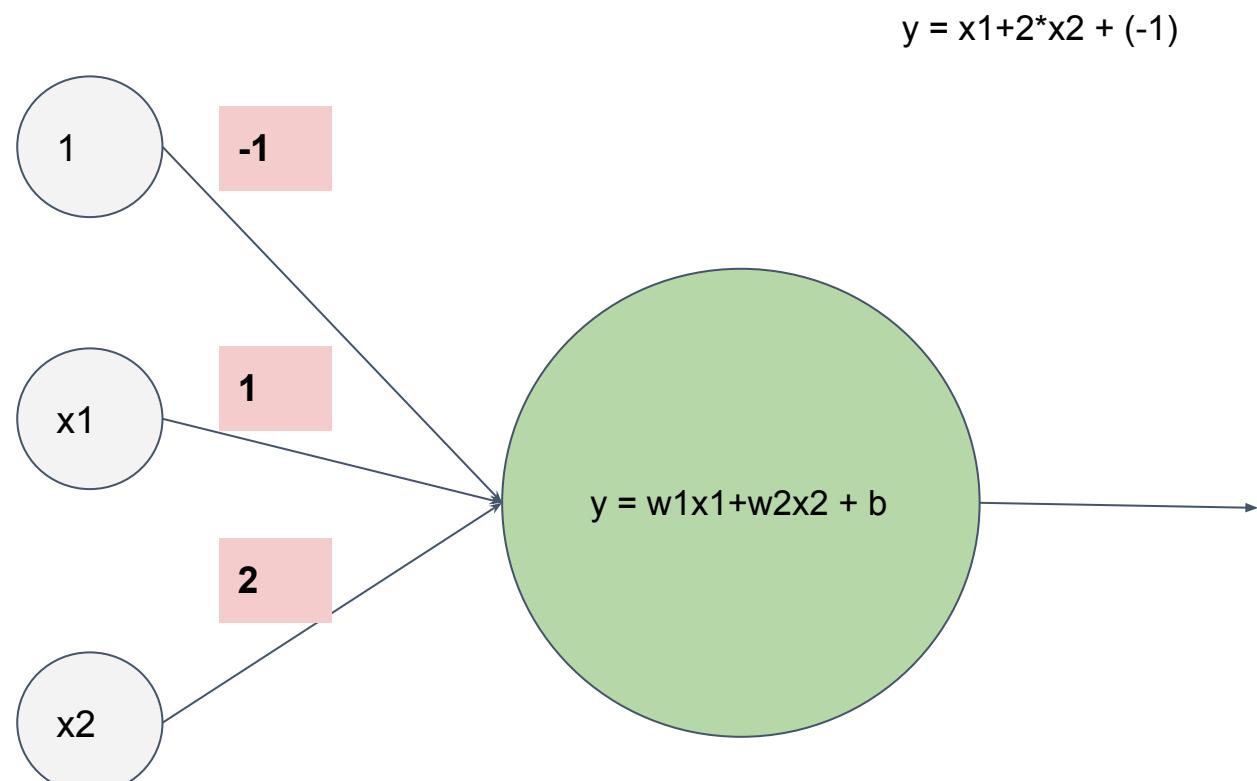
Deep Learning

Ejemplo práctico: ¿Qué sucede si todos nuestros pesos son 0?



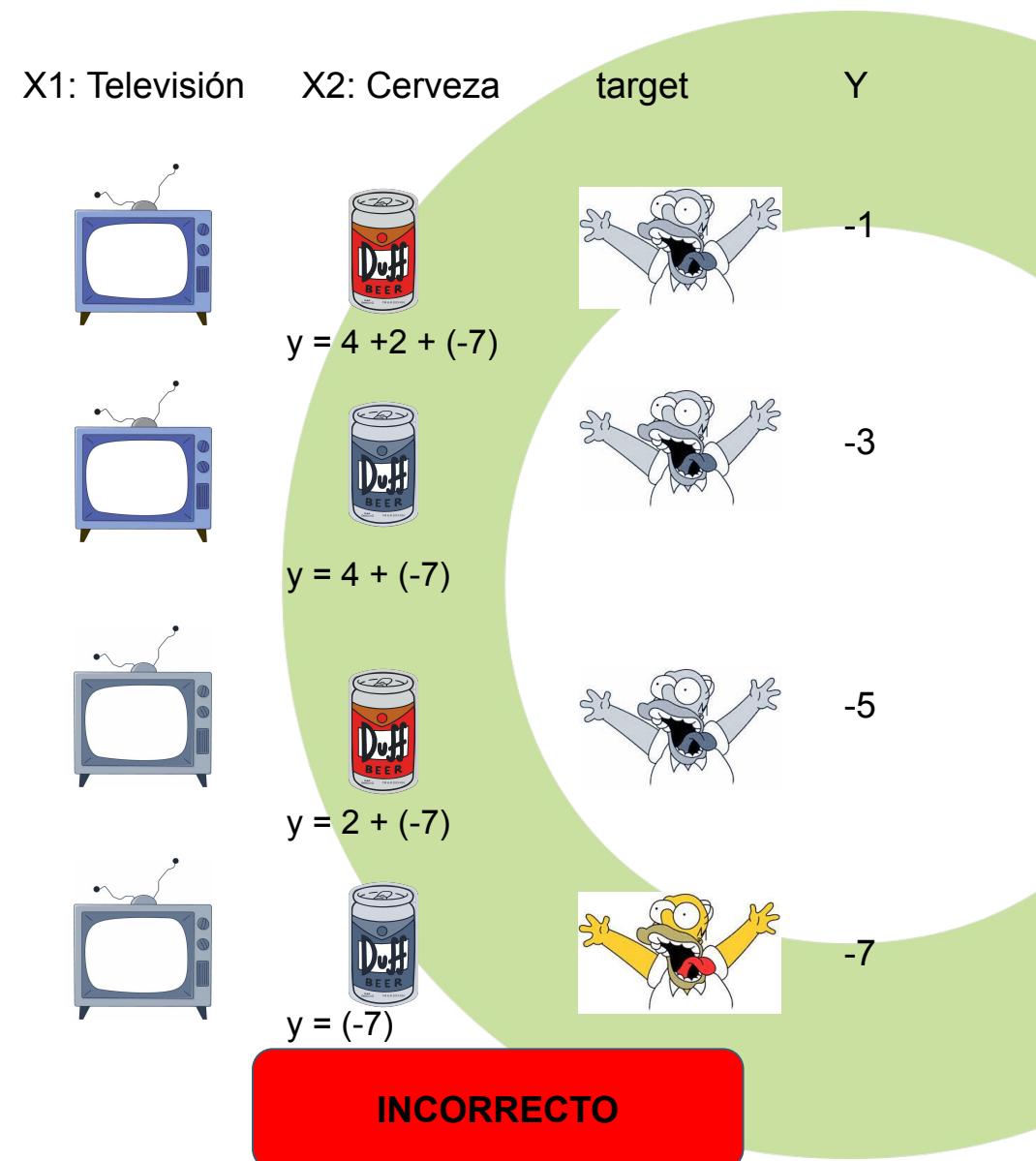
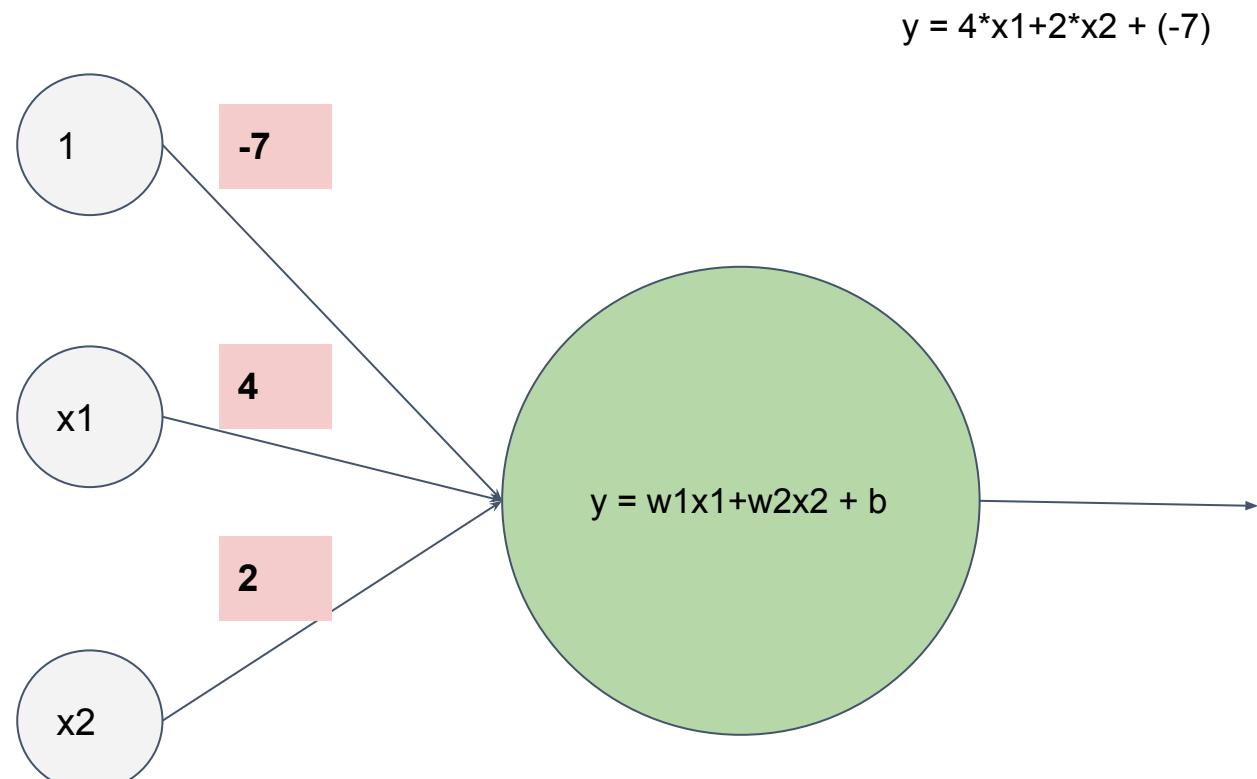
Deep Learning

Ejemplo práctico: Vamos a predecir si Homer pierde la cabeza.



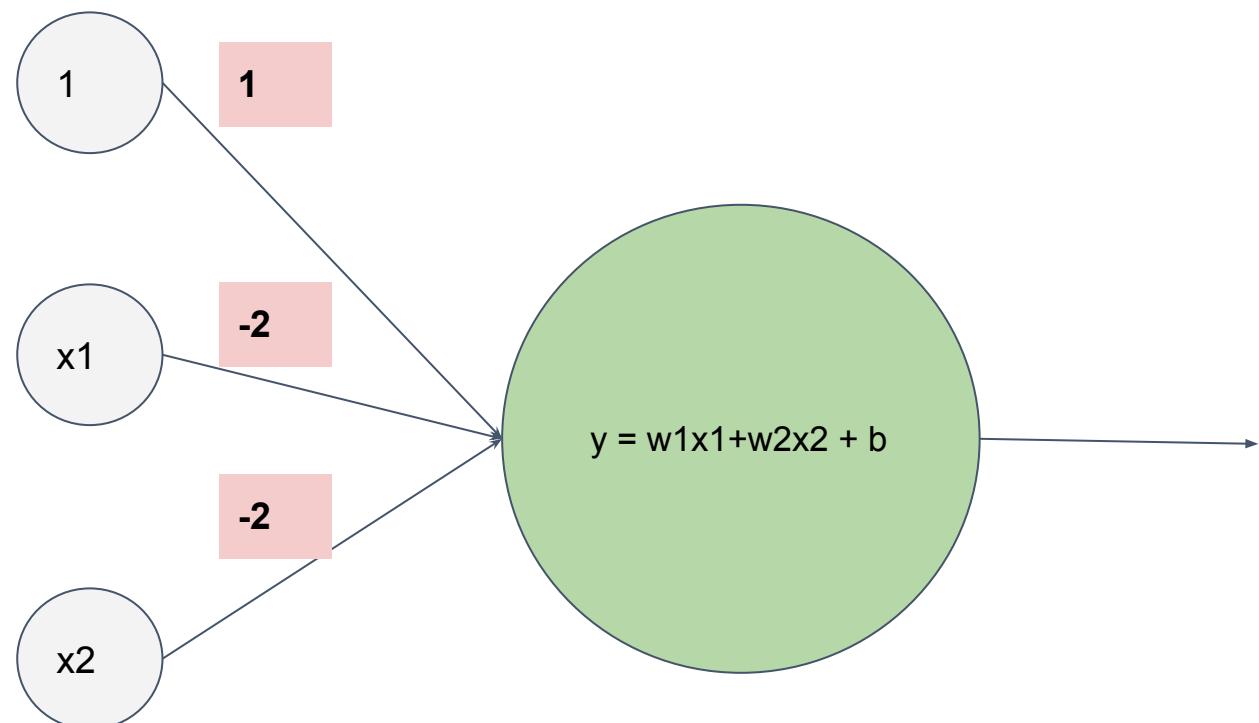
Deep Learning

Ejemplo práctico: Vamos a predecir si Homer pierde la cabeza.



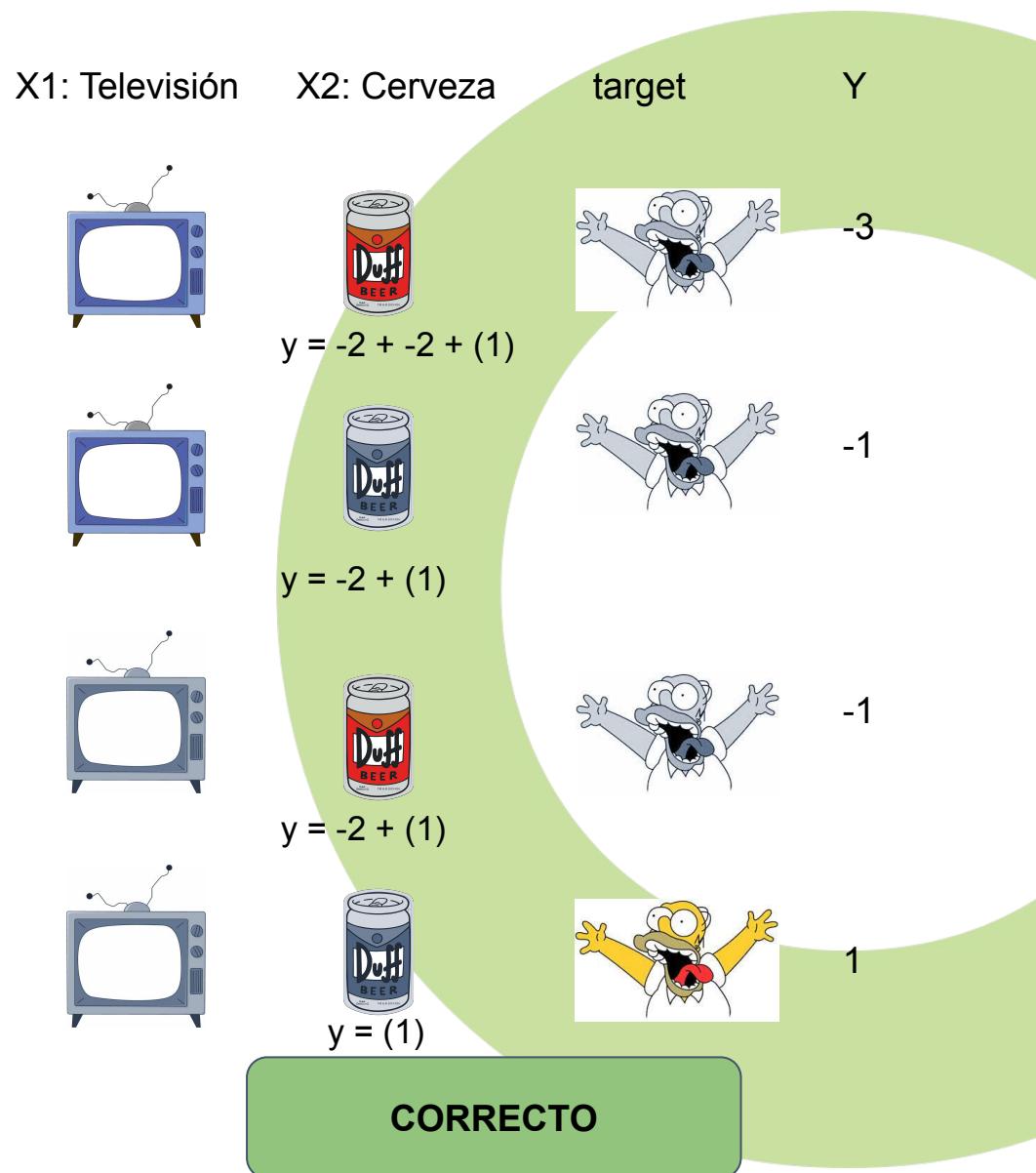
Deep Learning

Ejemplo práctico: Vamos a predecir si Homer pierde la cabeza.



$$y = (-2)*x_1+(-2)*x_2 + (1)$$

Por lo tanto, hemos encontrado una combinación de pesos para resolver nuestro caso de uso.

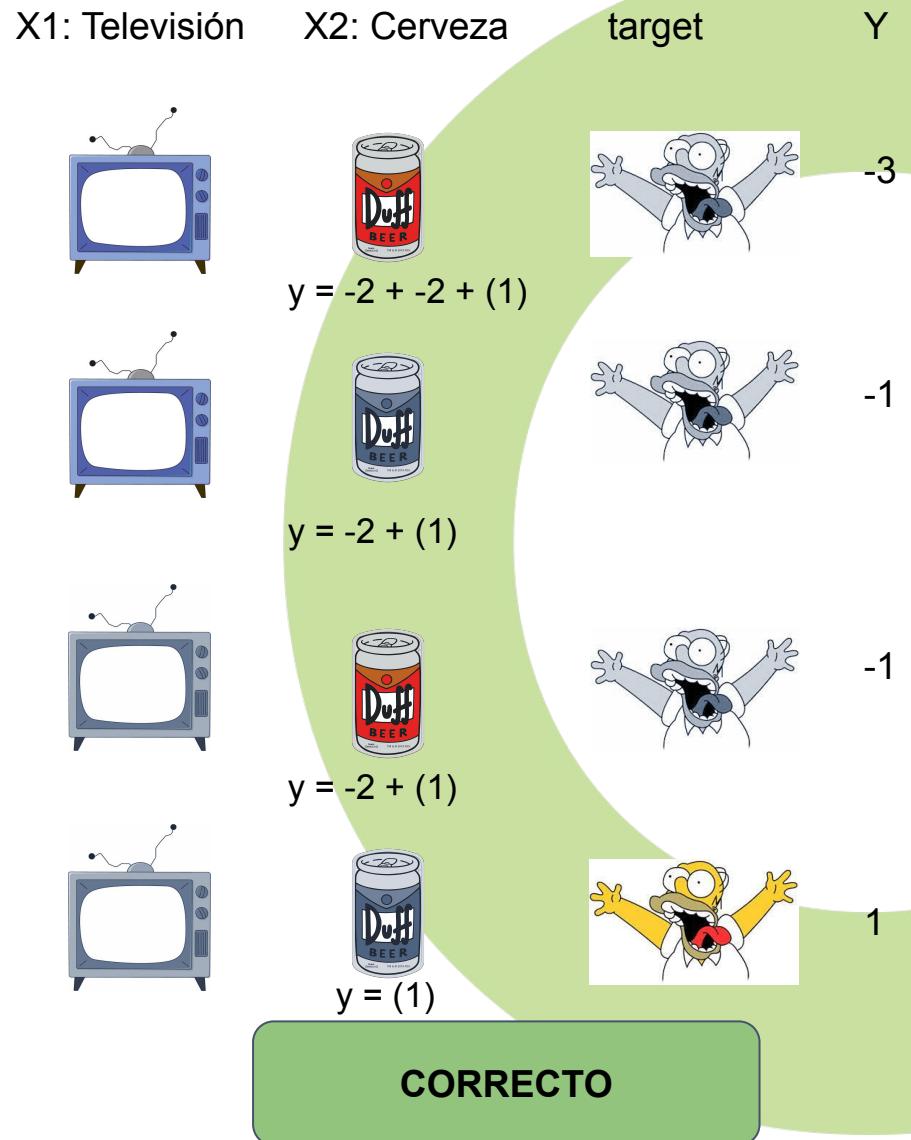


Deep Learning

Ejemplo práctico: Vamos a predecir si Homer pierde la cabeza.

x1	x2	y	result
1	1	-3	0
0	1	-1	0
1	0	-1	0
0	0	1	1

¿Os resulta similar a alguna función lógica este patrón?



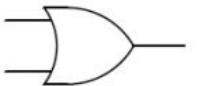
Deep Learning

Con esto por lo tanto, podemos construir funciones lógicas, tal que en función del valor de los pesos y la intersección, tenemos AND, OR, XOR, NOR.



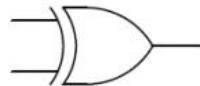
AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



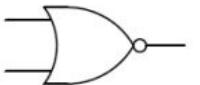
XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



NOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

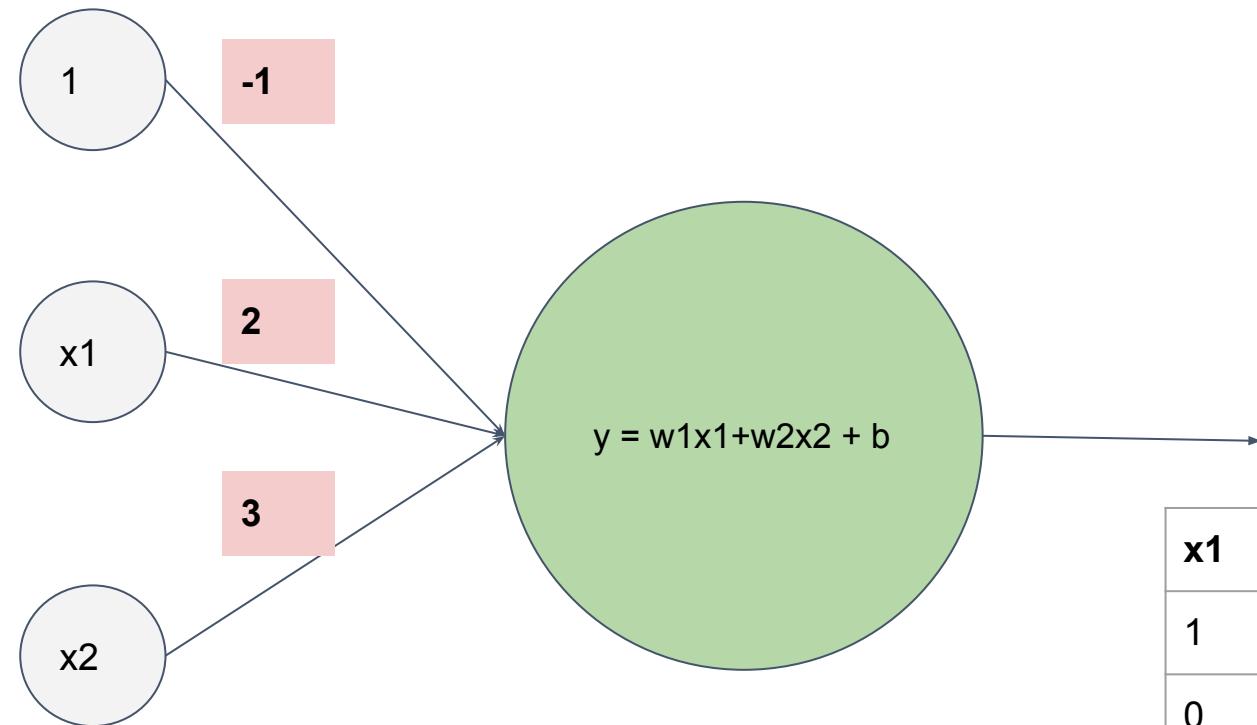


XNOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Deep Learning

Tenemos por tanto funciones lógicas, tal que en función del valor de los pesos y la intersección, tenemos AND, OR, XOR, NOR.



Pregunta

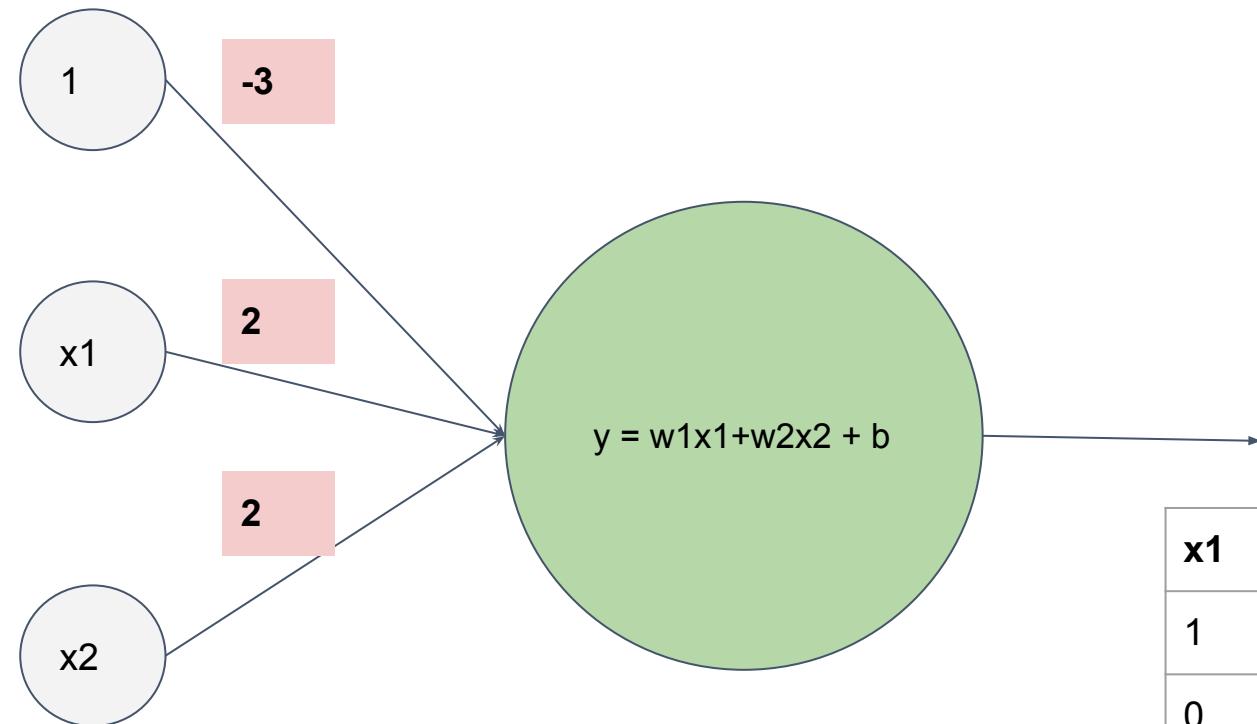
¿Qué es?

OR

x1	x2	y	result
1	1	4	1
0	1	2	1
1	0	1	1
0	0	-1	0

Deep Learning

Tenemos por tanto funciones lógicas, tal que en función del valor de los pesos y la intersección, tenemos AND, OR, XOR, NOR.



Pregunta

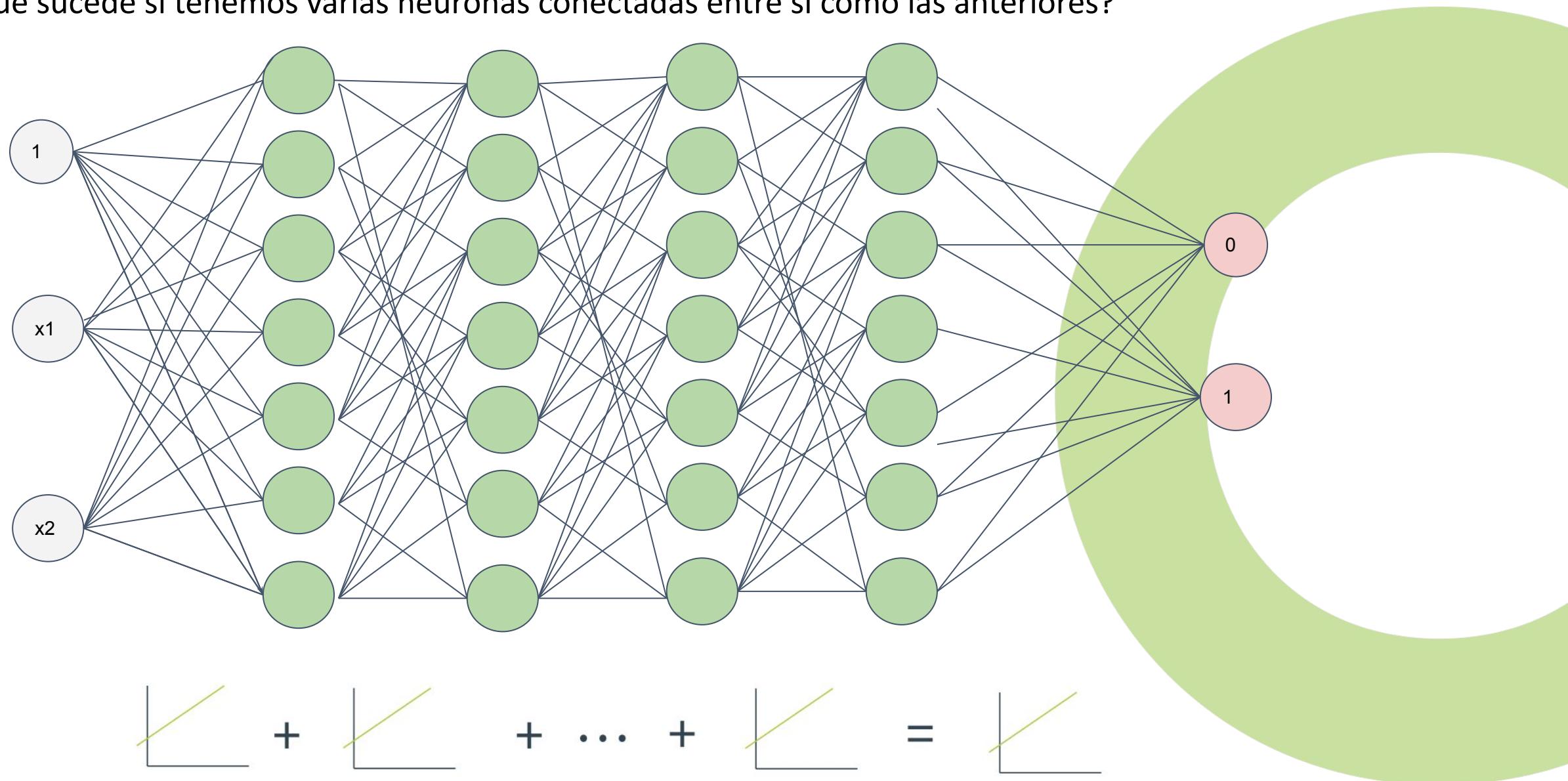
¿Qué es?

AND

x1	x2	y	result
1	1	1	1
0	1	-1	0
1	0	-1	0
0	0	-1	0

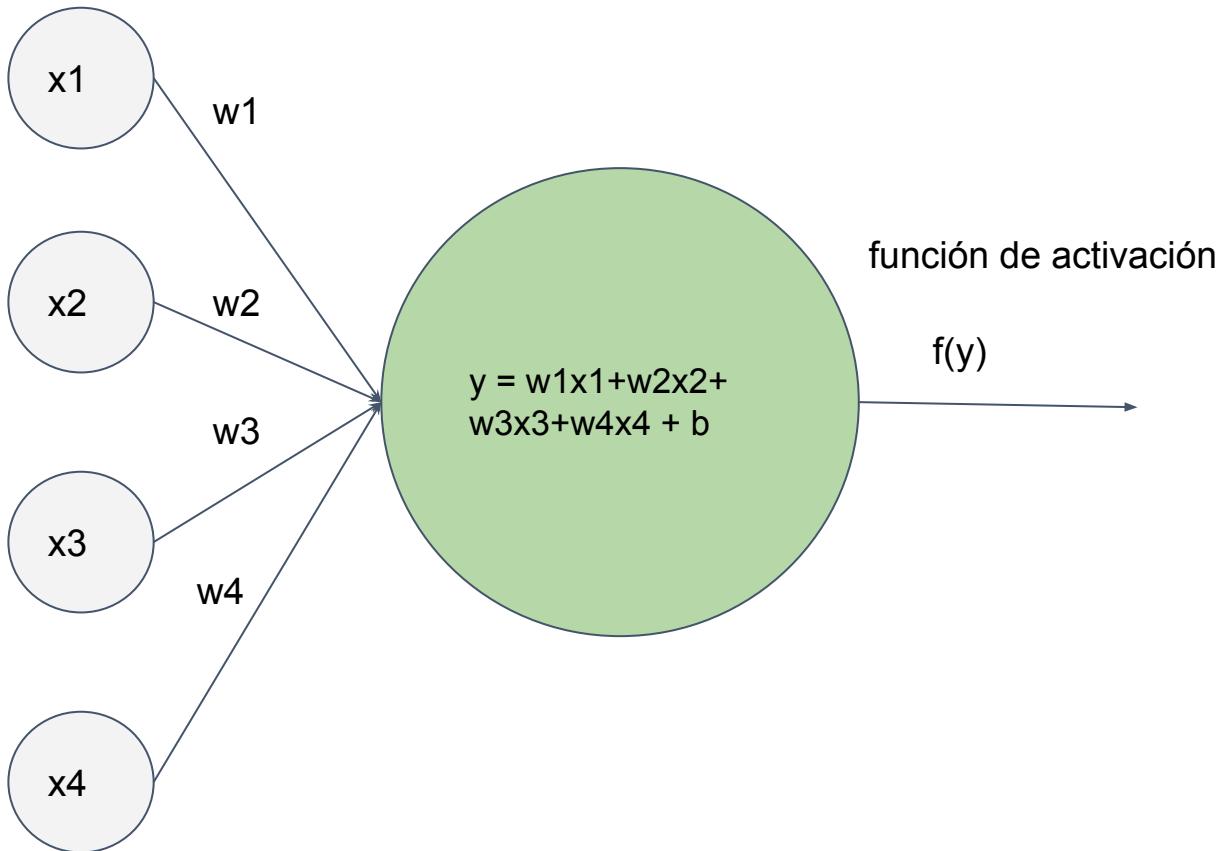
Funciones de activación

¿Qué sucede si tenemos varias neuronas conectadas entre sí como las anteriores?



Funciones de activación

Las funciones de activación permiten añadir no linealidad a los resultados.

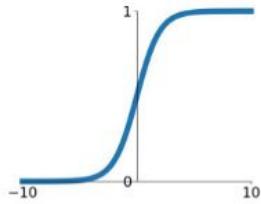


Funciones de activación

Tipos de funciones de activación

Sigmoid

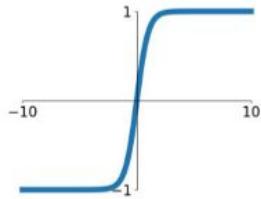
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Valores muy grandes saturan en "1" y los muy pequeños en "0".
Uso muy útil en la neurona de salida.

tanh

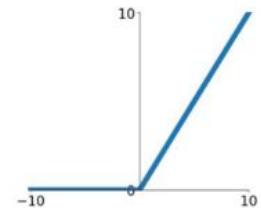
$$\tanh(x)$$



Misma forma que sigmoide pero entre -1 y 1. ▷ Prácticamente en desuso.

ReLU

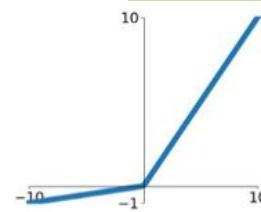
$$\max(0, x)$$



Muy utilizada en DL
▷ Evita vanishing gradient
▷ Acelera el tiempo de aprendizaje

Leaky ReLU

$$\max(0.1x, x)$$



Muy similar a la ReLU excepto por la parte negativa.

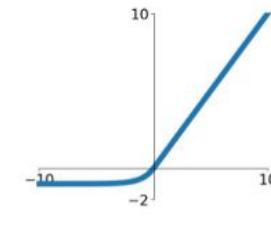
Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Máximo valor de salida

ELU

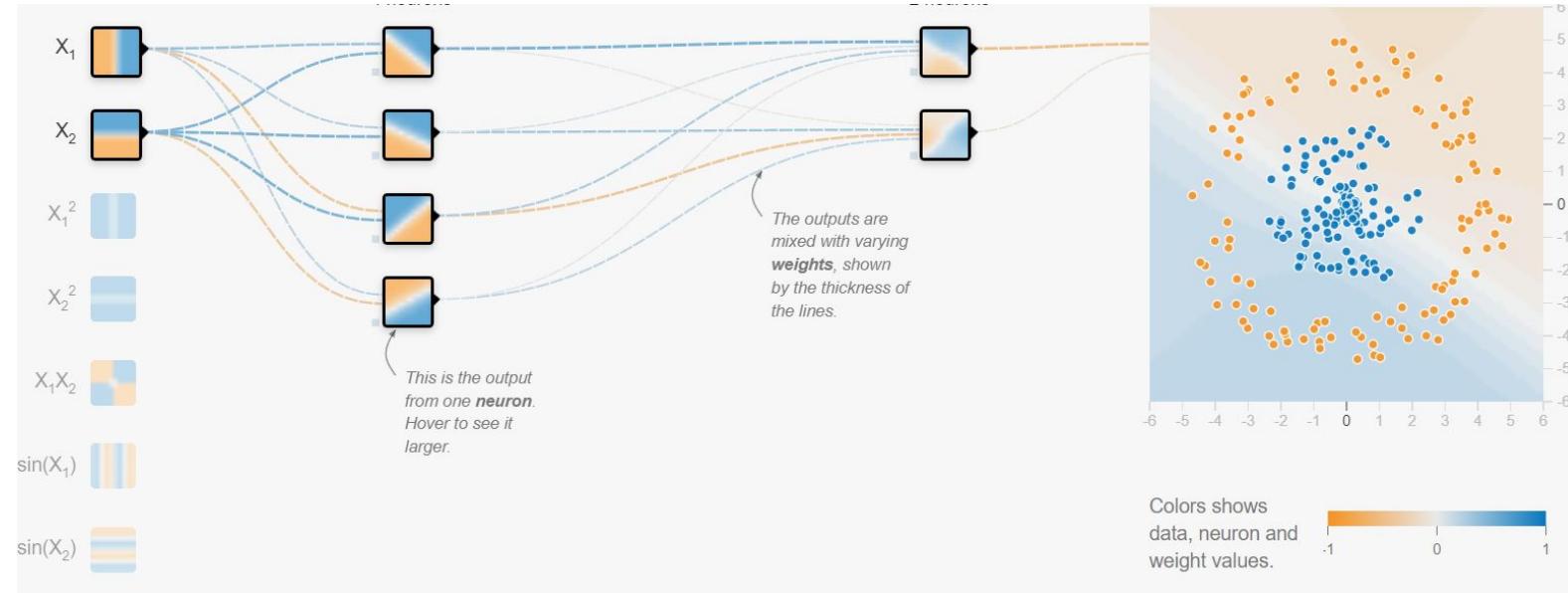
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Muy similar a la ReLU excepto por la parte negativa y la discontinuidad más suave.

Ejemplo

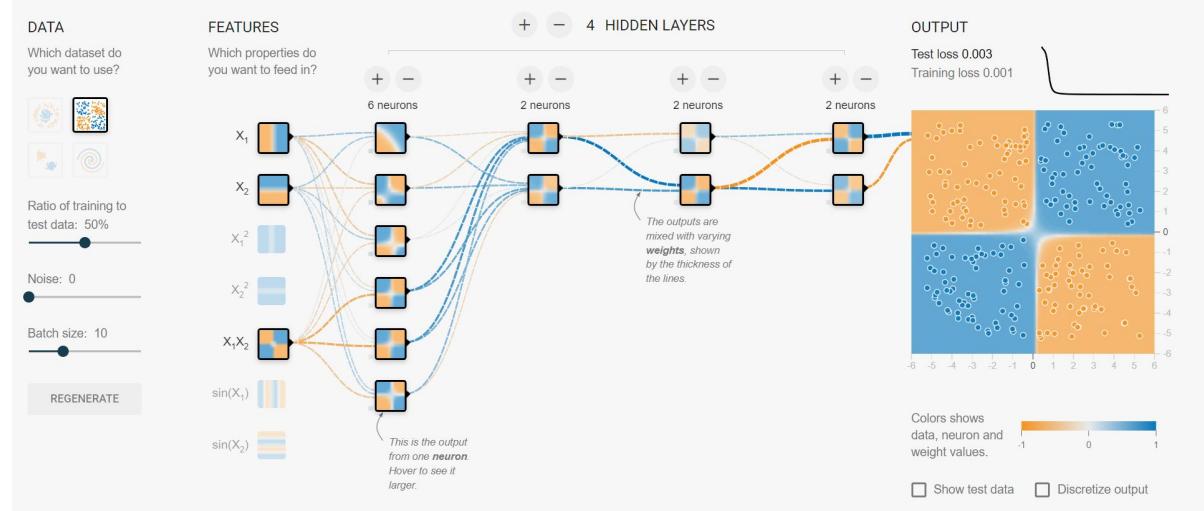
Ejemplo de red neuronal.



<https://playground.tensorflow.org/>

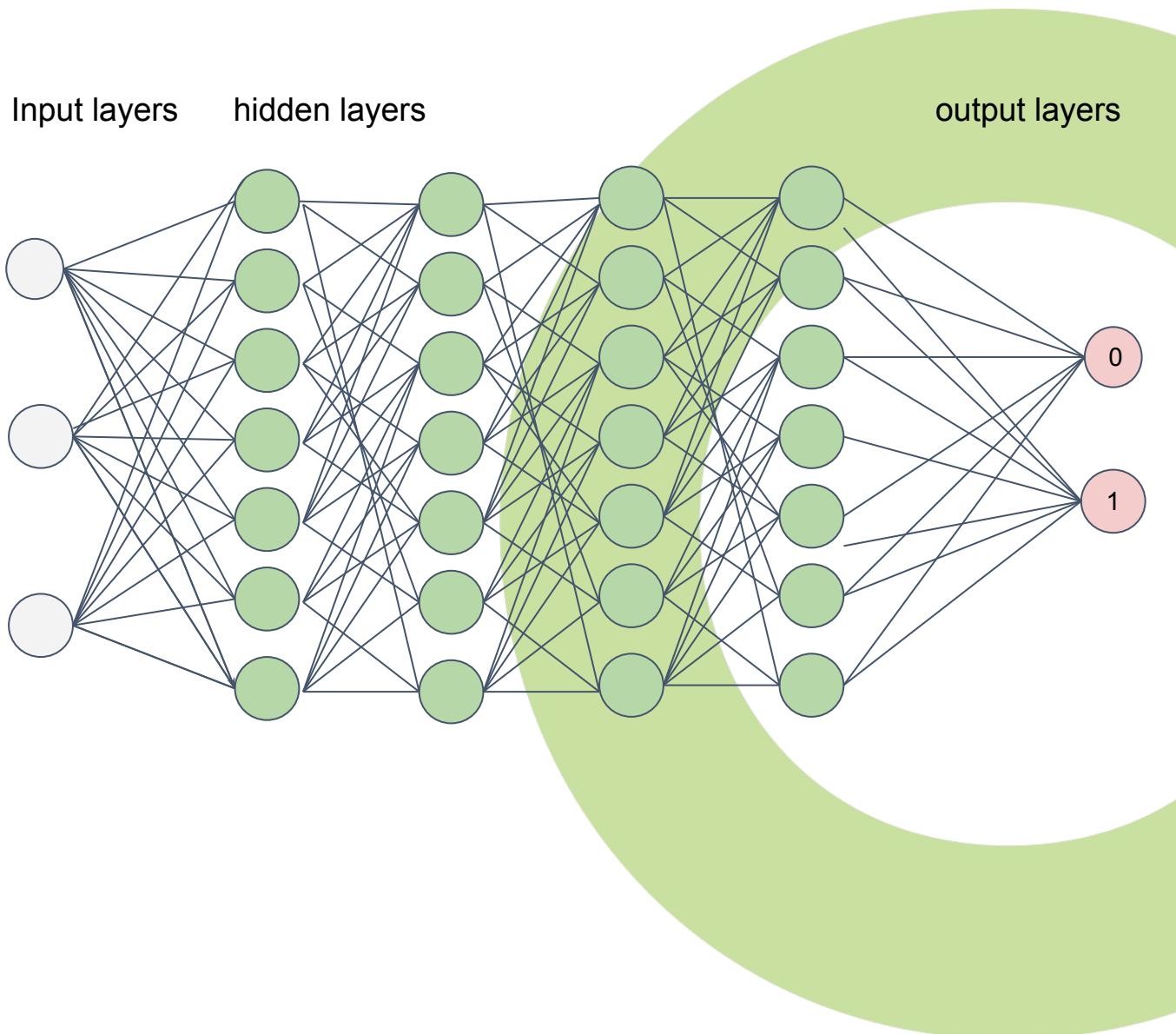
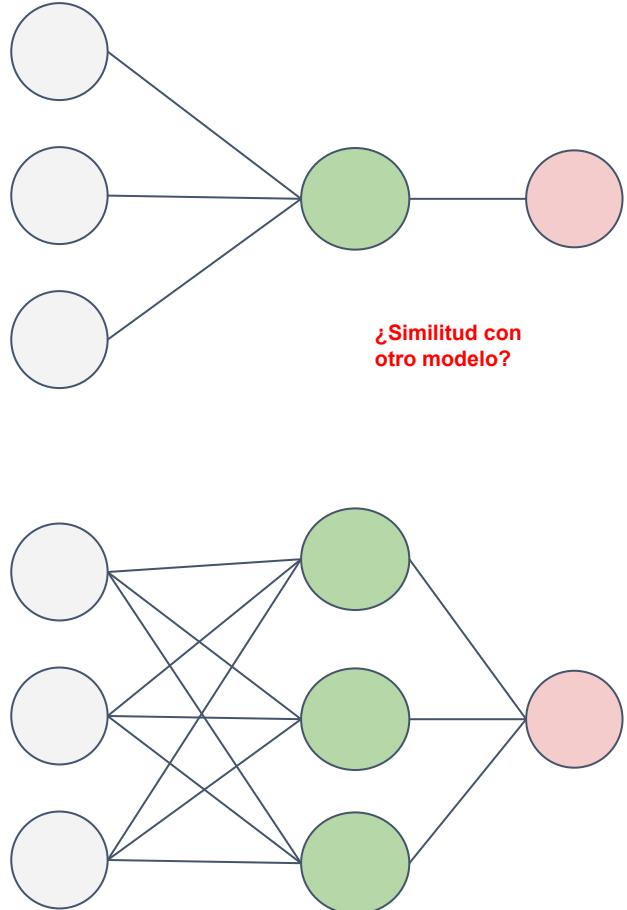
Ejercicio práctico con playground

- Manteniendo los valores por defecto de todo, prueba a reducir el número de redes neuronales de la primera capa intermedia a 1 y vete subiendo hasta que veas que el modelo es capaz de aprender correctamente. ¿Cuántas neuronas han sido necesarias para ello?
- Mantén los valores por defecto y pon 5 neuronas en la misma capa. Cambia la función de activación de tanh a linear. ¿Qué sucede?
- Reinicia la página y pon en este caso la espiral a predecir. ¿Es el modelo por defecto capaz de predecir correctamente la misma? Pon 4 capas intermedias con 6 neuronas cada una.
 - Prueba ahora a añadir otras variables de entrada y ver si mejoran los resultados.
 - Manteniendo todas las variables de entrada, pon una única capa de 6 neuronas. ¿Qué observas?



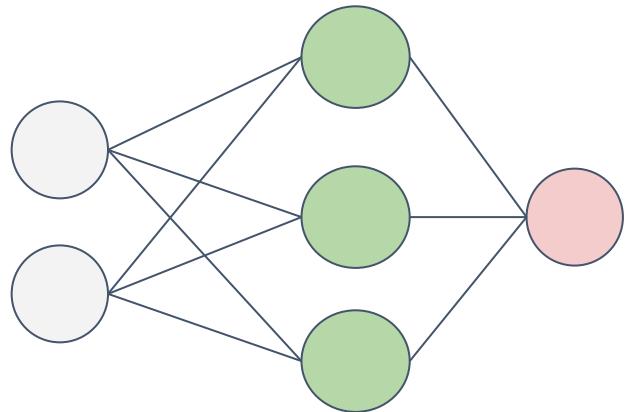
Redes neuronales

Ejemplos de redes neuronales.



Redes neuronales

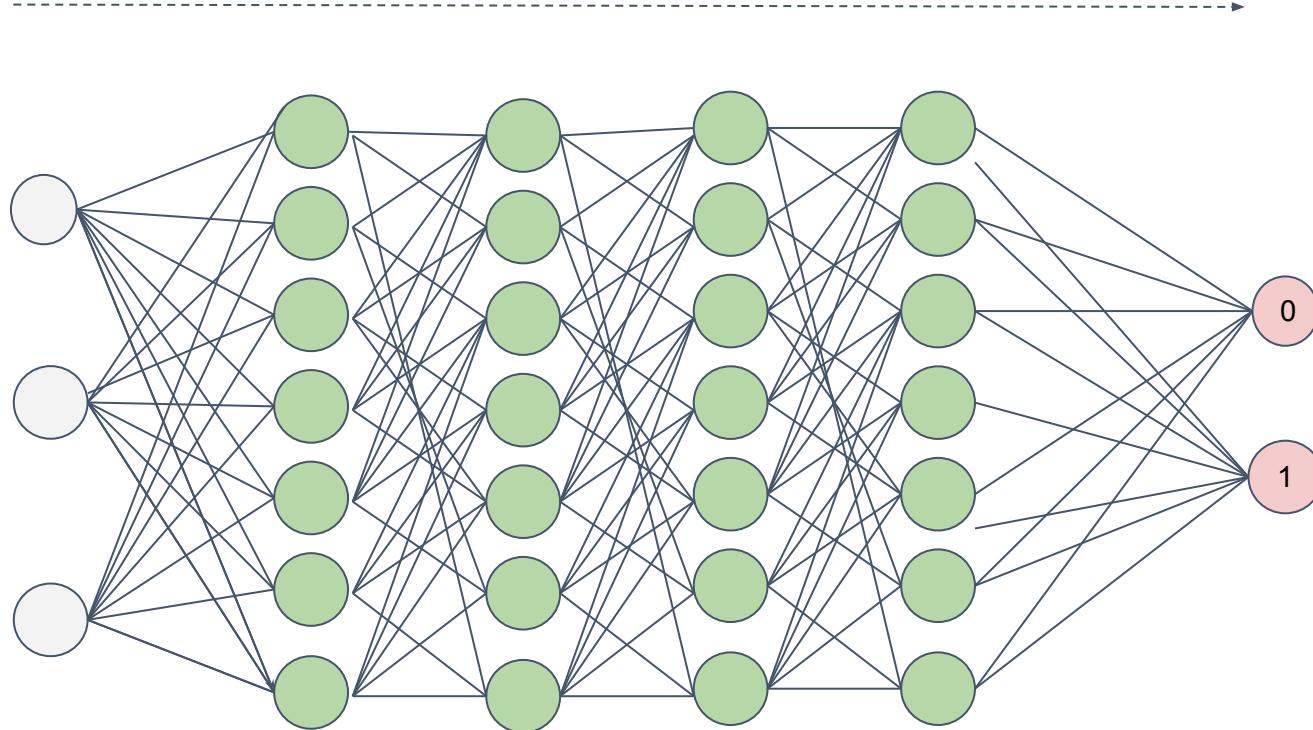
Las redes neuronales son operaciones matriciales.



$$\begin{bmatrix} y_1 = \sigma(w_{11}x_1 + w_{21}x_2 + b_1) \\ y_2 = \sigma(w_{12}x_1 + w_{22}x_2 + b_2) \\ y_3 = \sigma(w_{13}x_1 + w_{23}x_2 + b_3) \end{bmatrix} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

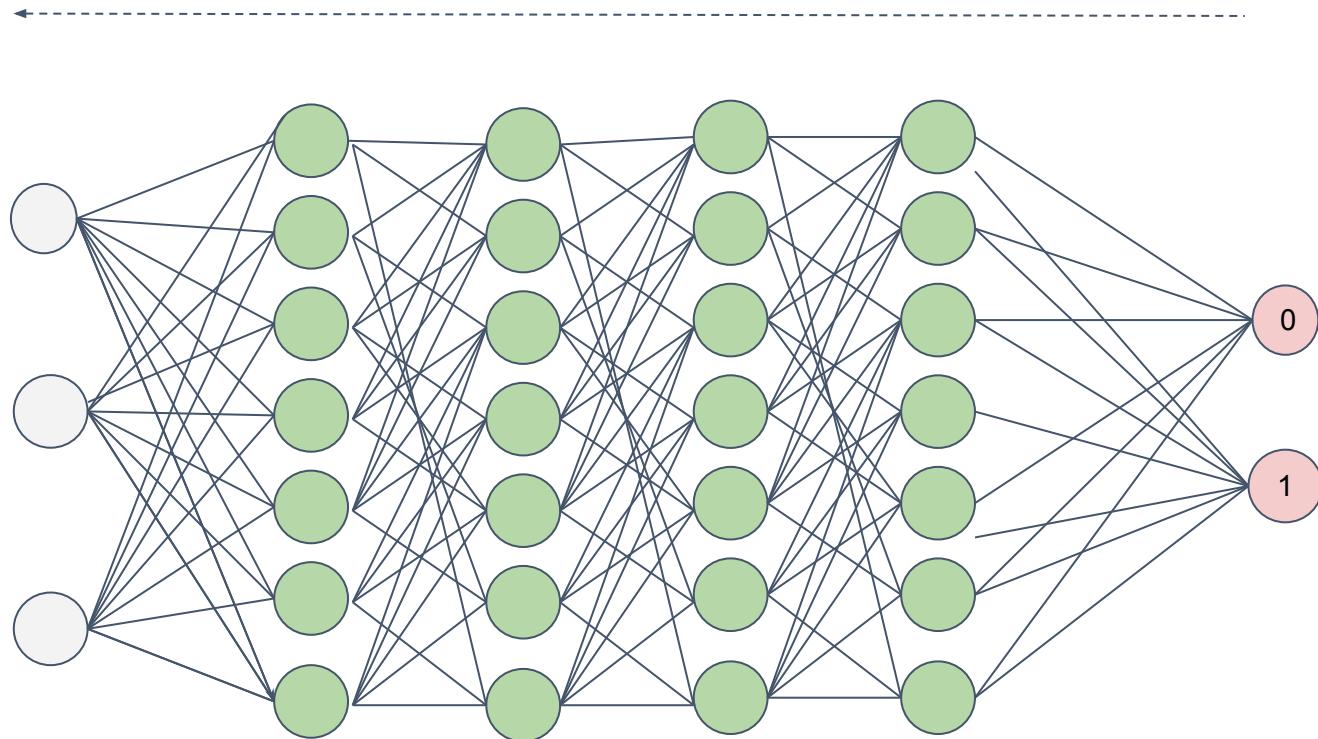
Ejemplo

Proceso de aprendizaje: Forward pass. (Proceso también de predicción)



Ejemplo

Backward pass: Mejora de los pesos de las redes en el entrenamiento.

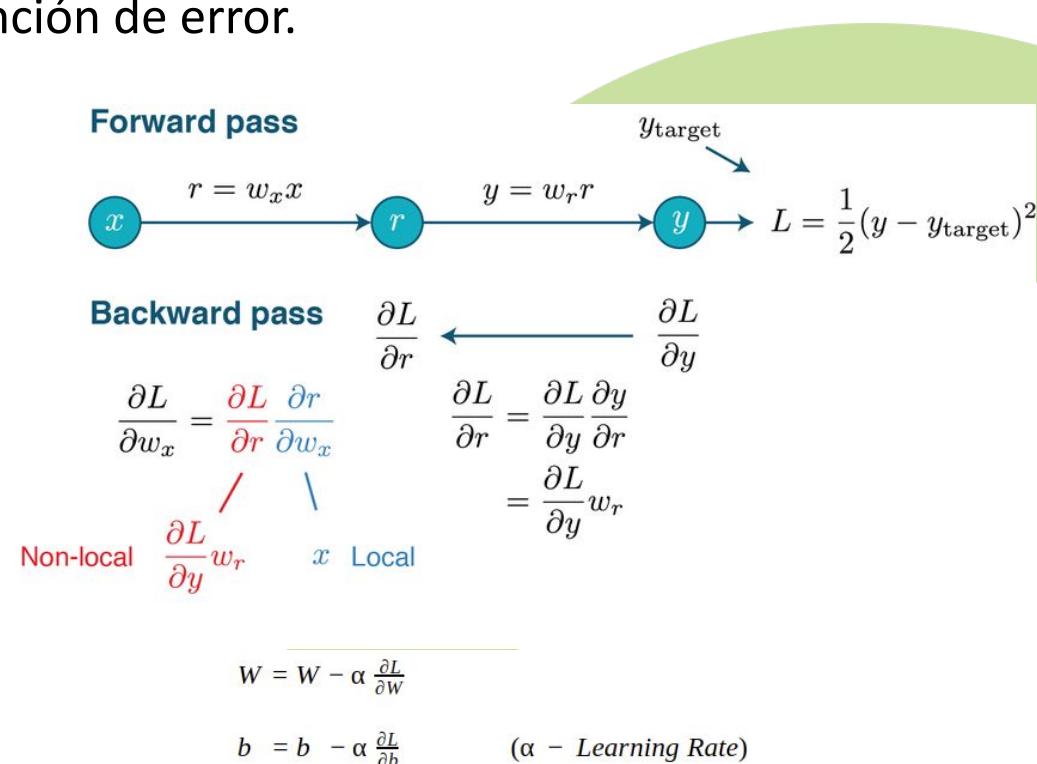


http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Funcionamiento del backward pass

Backward pass: Propagación del cálculo del gradiente dada la función de error.

1. Introducción de un mini-lote con N muestras aleatorias a la entrada de la red.
2. Se realiza el forward pass pasando los datos por todas las neuronas y activando aquellas según sus pesos.
3. Llegado al final, la output layer, se hace una evaluación de función de coste.
4. Cálculo del gradiente (derivada multivariable de la función de coste respecto a los parámetros de la red).
5. Actualización de los parámetros de la red restando el gradiente a su valor actual y multiplicado por la tasa de aprendizaje (learning rate)
6. Iterar durante diferentes epochs(ciclo del algoritmo en el que la red ve todas las muestras disponibles una vez) evitando sobreajuste.



Possible problemas:

- Es posible acabar en un mínimo local o en un punto silla.
- Es necesario encontrar el learning rate adecuado:
 - Valor no muy alto para evitar oscilaciones en torno a un mínimo local
 - Valor no muy bajo para evitar que tarde mucho en llegar al mínimo.
 - Puede ser fija o variable con el tiempo.
 - Puede ser la misma para toda la red, o distinta por capa o grupos de capas.

Índice de la sesión

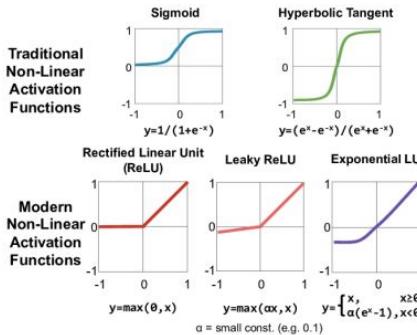
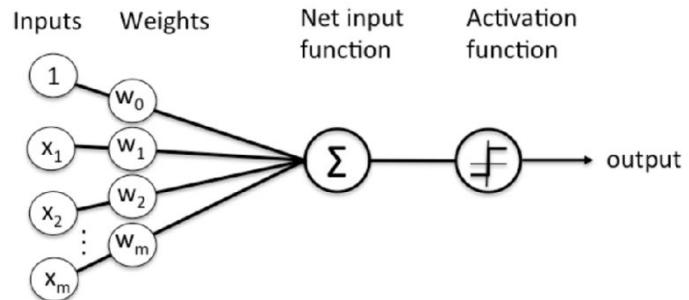
- Recap
- Introducción a las redes neuronales
- **Configuración de los modelos de redes neuronales**
- Modelos de Deep Learning



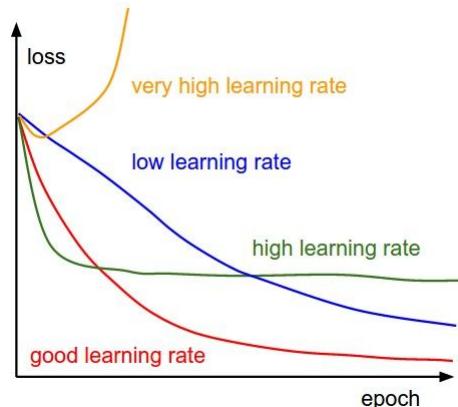
Redes neuronales

Puntos importantes en la definición de las redes neuronales.

Activation function



Learning rate



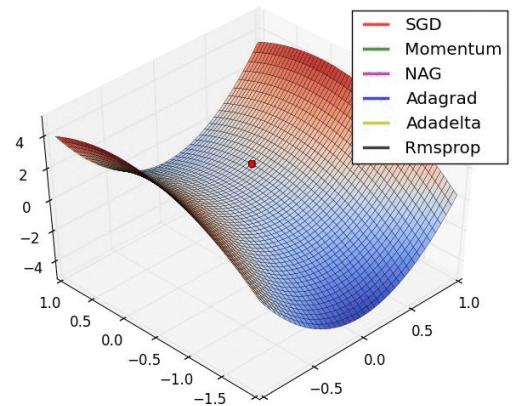
Batch Size & Epochs

- **Batch Size:** Número de observaciones que pasan por cada entrenamiento de la red.
- **Epochs:** Número de veces por el que la red pasan todos los datos de entrenamiento.

Redes neuronales

Puntos importantes en la definición de las redes neuronales.

Optimizers



• Regresión

- MAE
- MSE

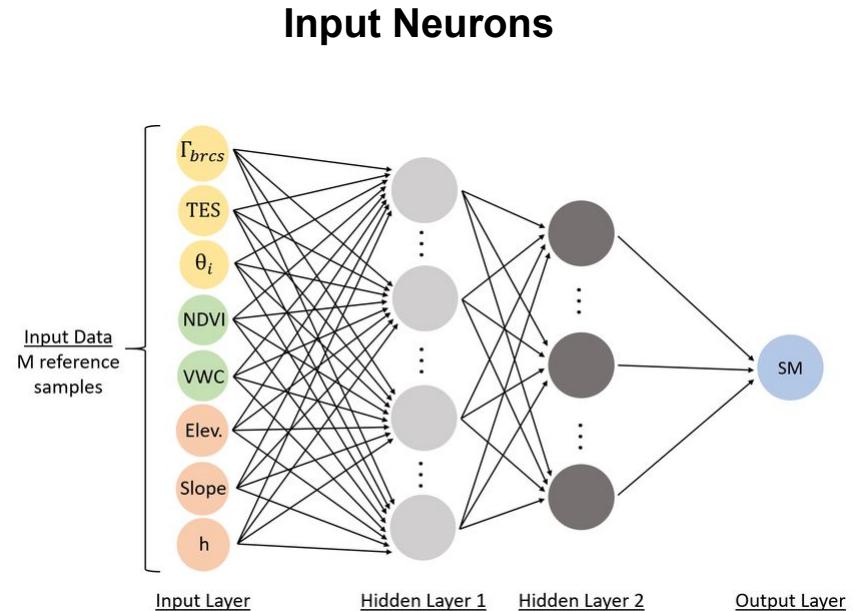
• Clasificación

- Cross-entropy
- Binary Cross Entropy

Loss function

Redes neuronales

Puntos importantes en la definición de las redes neuronales.



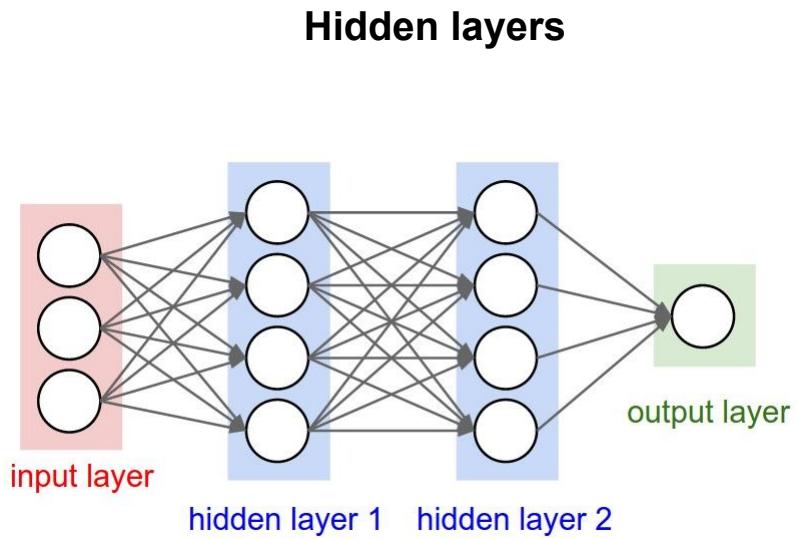
El número de perceptrones ha de ser igual al número de features de entrada.

Output Neurons

- **Regresión:** Una neurona por valor predicho
- **Clasificación:**
 - Para clasificación binaria una neurona por clase positiva.
 - Multi-class: Un perceptrón por clase.

Redes neuronales

Puntos importantes en la definición de las redes neuronales.



Hidden Neurons

- En general es bueno usar el mismo número de perceptrones en todas las capas
- Es mejor añadir capas que perceptrones.

Es dependiente del problema. La idea es probar para que no sea demasiado grande ni pequeño:

- Dense/Fully connected
- Convolutional
- Max-pooling
- flatten

Ejemplo en keras

Proceso de desarrollo de una red neuronal con Keras.

1. Modelo
Input layer
Hidden layers
Output layer

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

# Añadimos una capa densa
model.add(Dense( 2, input_shape=(3,)),activation="relu")
# Añadimos una única neurona final para la predicción
model.add(Dense( 1 ))

# resumen
model.summary()
```

Ejemplo en keras

Proceso de desarrollo de una red neuronal con Keras.

1. Modelo
Input layer
Hidden layers
Output layer

2. Compilador
Optimizador
Función de pérdida

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

# Añadimos una capa densa
model.add(Dense( 2, input_shape=(3,)),activation="relu")
# Añadimos una única neurona final para la predicción
model.add(Dense( 1 ))

# resumen
model.summary()

model.compile(optimizer="adam", loss="mse")
```

Ejemplo en keras

Proceso de desarrollo de una red neuronal con Keras.

1. Modelo

Input layer
Hidden layers
Output layer

2. Compilador

Optimizador
Función de pérdida

3. Entrenar

Epochs
Batch size

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

# Añadimos una capa densa
model.add(Dense( 2, input_shape=(3,)),activation="relu")
# Añadimos una única neurona final para la predicción
model.add(Dense( 1 ))

# resumen
model.summary()

model.compile(optimizer="adam", loss="mse")

model.fit(X_train, y_train, epochs=5)
```

Ejemplo en keras

Proceso de desarrollo de una red neuronal con Keras.

1. Modelo

Input layer
Hidden layers
Output layer

2. Compilador

Optimizador
Función de pérdida

3. Entrenar

Epochs
Batch size

4. Predecir y evaluar

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

# Añadimos una capa densa
model.add(Dense( 2, input_shape=(3,)),activation="relu")
# Añadimos una única neurona final para la predicción
model.add(Dense( 1 ))

# resumen
model.summary()

model.compile(optimizer="adam", loss="mse")

model.fit(X_train, y_train, epochs=5)

preds = model.predict(X_test)

model.evaluate(X_test, y_test)
```

Ejercicio práctico

Haz un modelo de clasificación usando mnist.

```
from tensorflow.keras.datasets import mnist  
  
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

https://colab.research.google.com/drive/1dQurmQhuUA-jLHTqSlkYyPUVb7sXBm2_#scrollTo=DzpyRq4BPs9Q

Tips:

1. Normaliza los datos dividiendo entre 255.
2. Crea un modelo similar al siguiente

```
model.summary()  
  
Model: "sequential"  
-----  
Layer (type)          Output Shape         Param #  
-----  
flatten (Flatten)     (None, 784)           0  
-----  
dense (Dense)         (None, 300)           235500  
-----  
dense_1 (Dense)       (None, 100)           30100  
-----  
dense_2 (Dense)       (None, 10)            1010  
-----  
Total params: 266,610  
Trainable params: 266,610  
Non-trainable params: 0
```

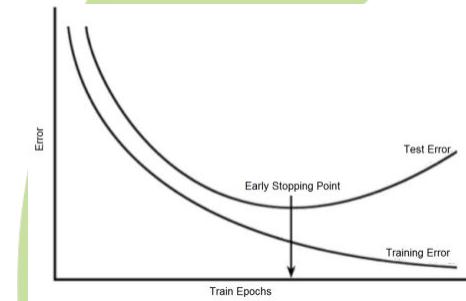
Redes neuronales

Puntos importantes en la definición de las redes neuronales.

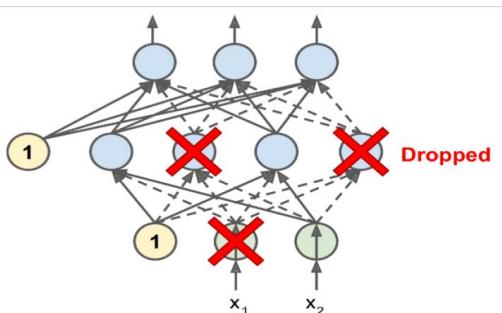
Overfitting

- Regularización L1, L2
- Dropout
- Data augmentation
- Early stopping

Early stopping

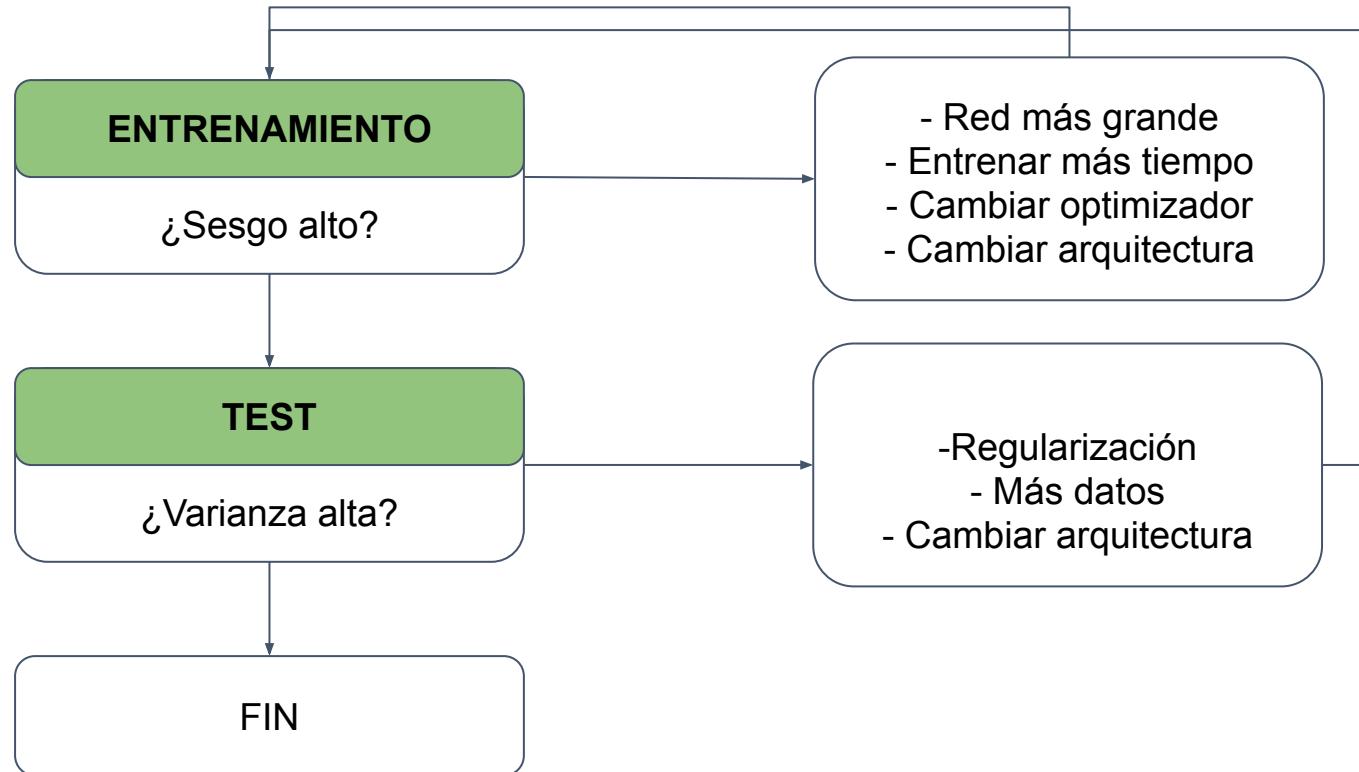


Drop Out



Modelos de Deep Learning

Pasos para entrenar una red neuronal profunda.



Regularización

Métodos que permiten prevenir el sobreajuste de los modelos.

- Regularización L1, L2
- Dropout
- Data augmentation
- Early stopping

Regularización

L1 y L2 regularización.

L1 Regularization

$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j|$$

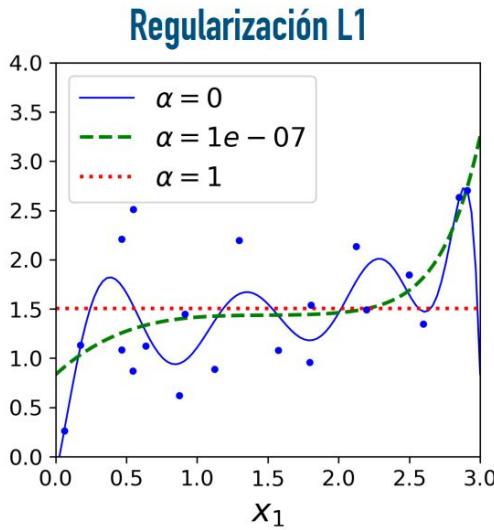
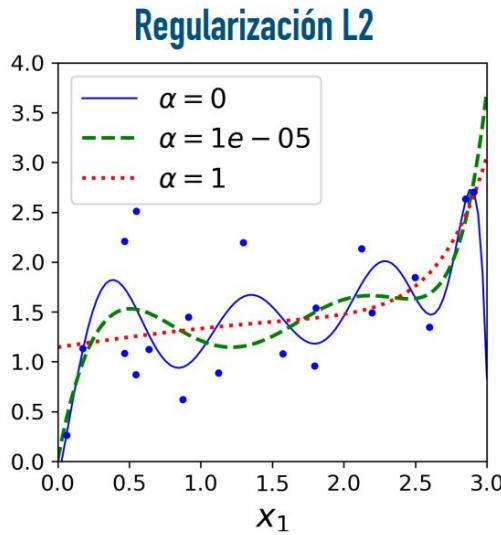
L2 Regularization

$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M W_j^2$$

Loss function Regularization Term

Regularización

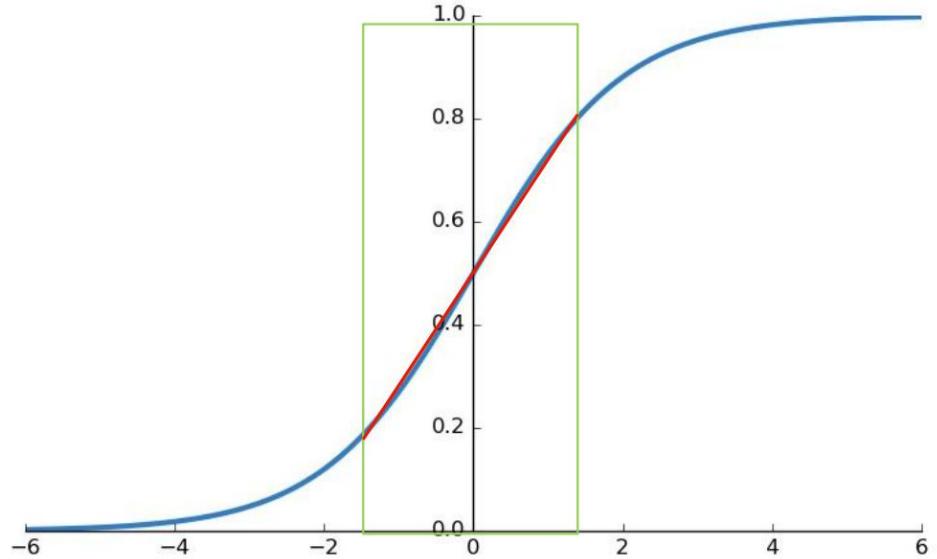
L1 y L2 regularización.



- L2 y L1 previenen que los pesos tomen valores altos.
- L1 tiende a eliminar completamente los pesos de las características menos importantes.

Regularización

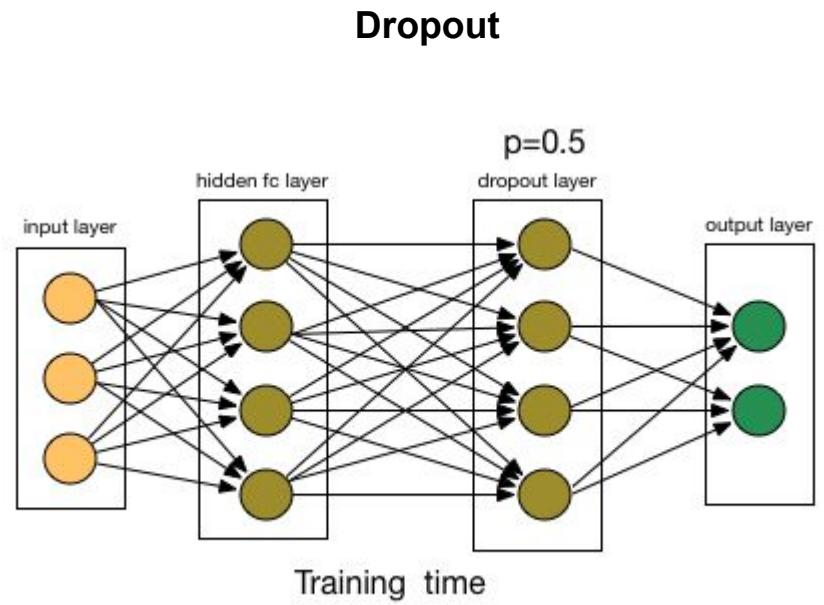
Métodos que permiten prevenir el sobreajuste de los modelos.



- Trabajamos en la zona lineal de la sigmoide.
- Al no aplicar “no linealidades” el modelo no sobre-ajusta

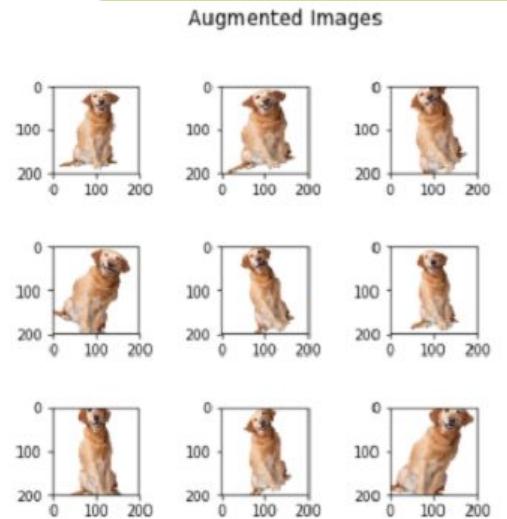
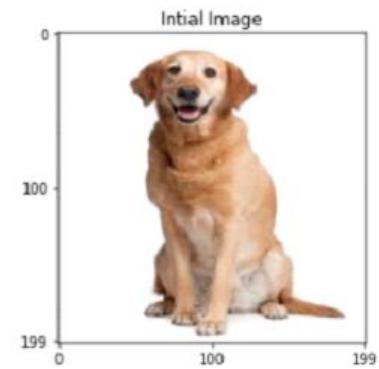
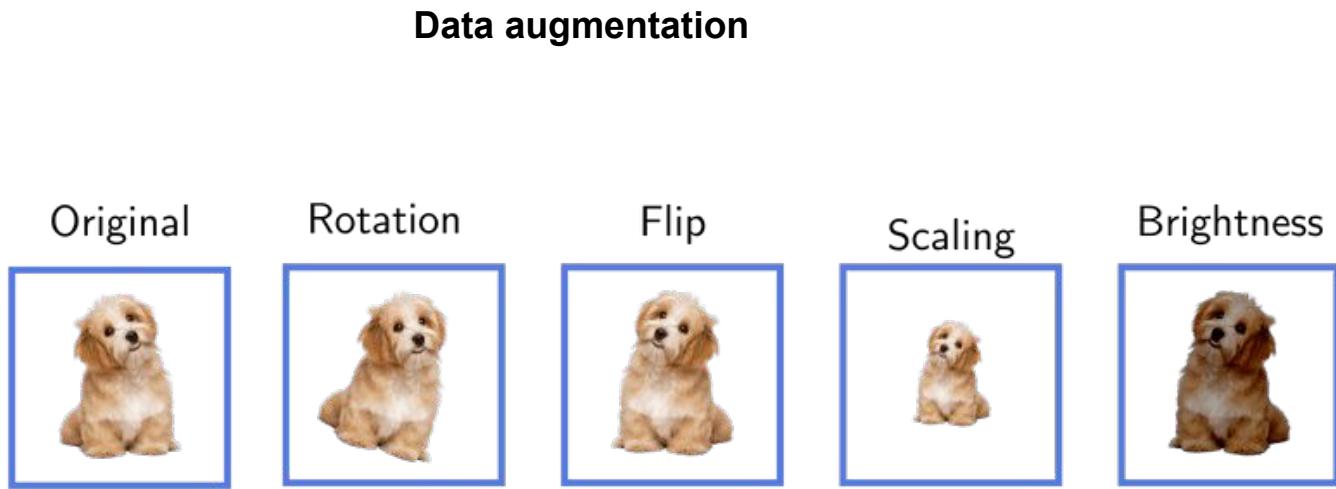
Regularización

Dropout



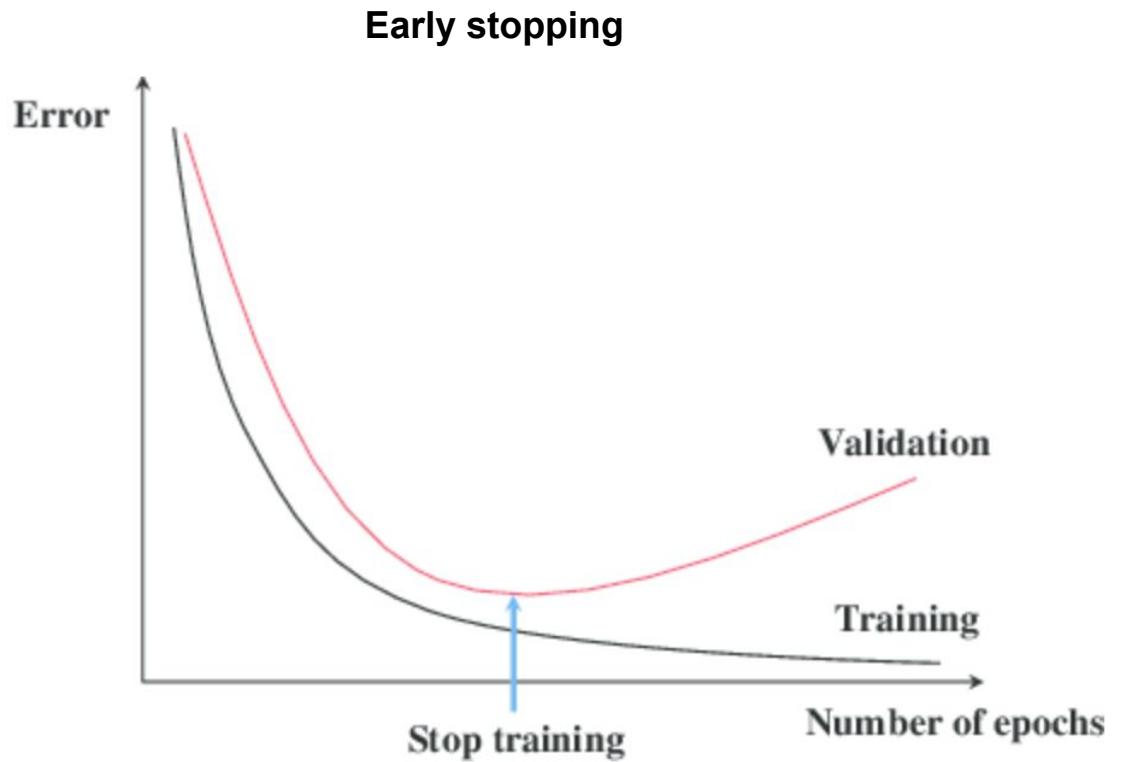
Regularización

Data Augmentation



Regularización

Early stopping



Ejercicio práctico

Del modelo de MNIST anterior, crea otro nuevo modelo que entre cada capa un dropout(0.5) y evalúa los resultados. ¿Qué le sucede a los resultados?

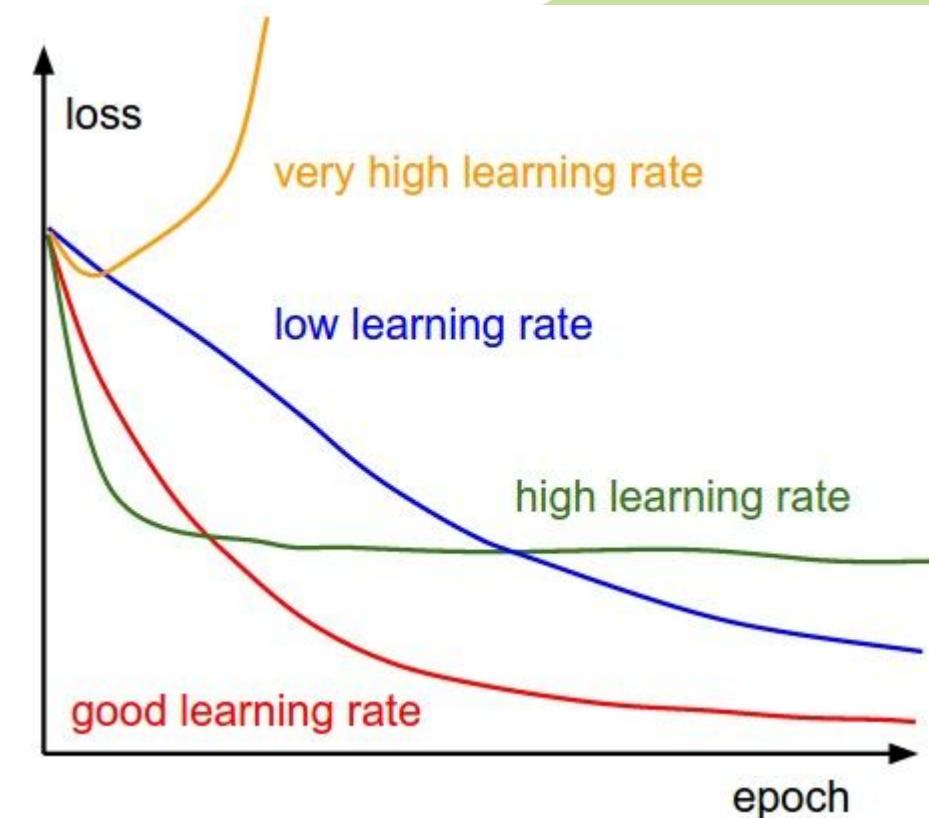
```
from tensorflow.keras.datasets import mnist  
  
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

https://colab.research.google.com/drive/1dQurmQhuUA-jLHTqSIkYyPUVb7sXBm2_#scrollTo=DzpyRq4BPs9Q

Problemas de convergencia

Es importante trabajar métodos que permitan encontrar la convergencia del modelo a la solución óptima.

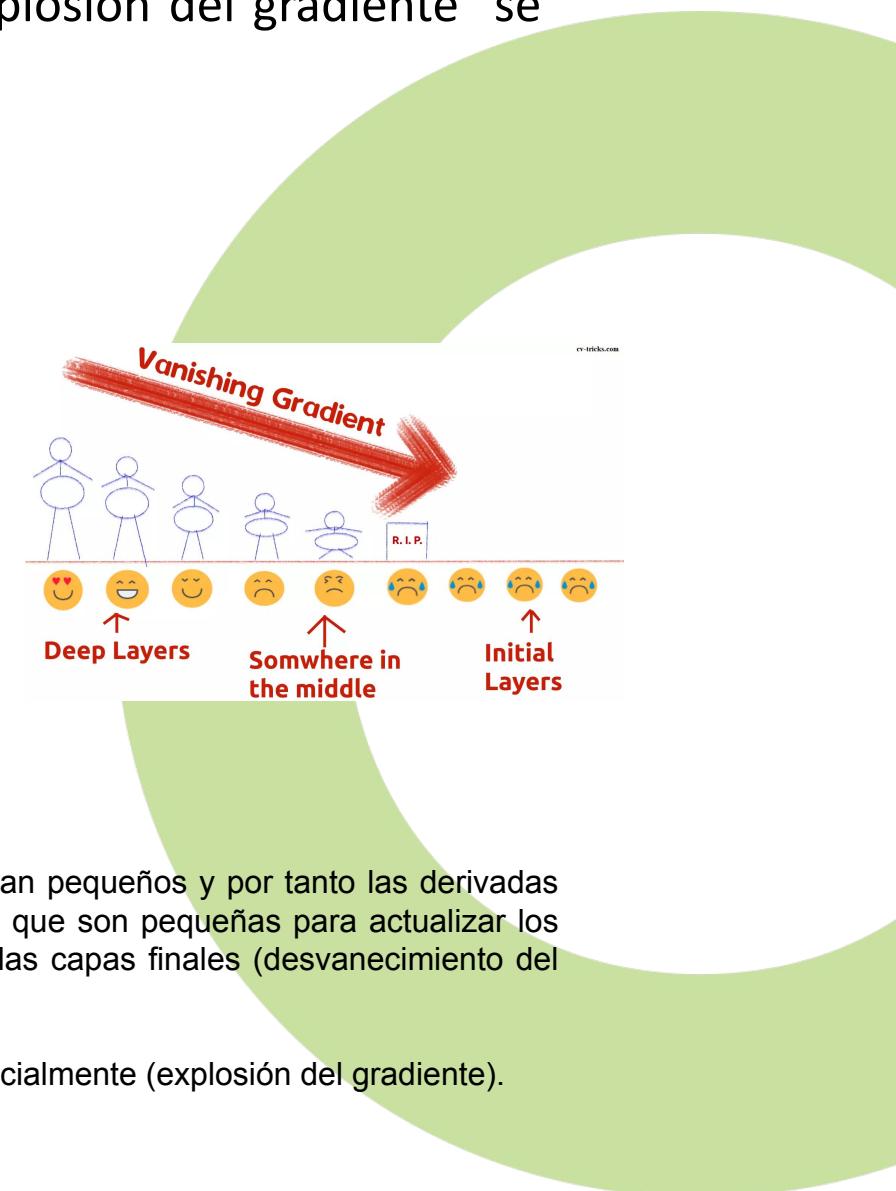
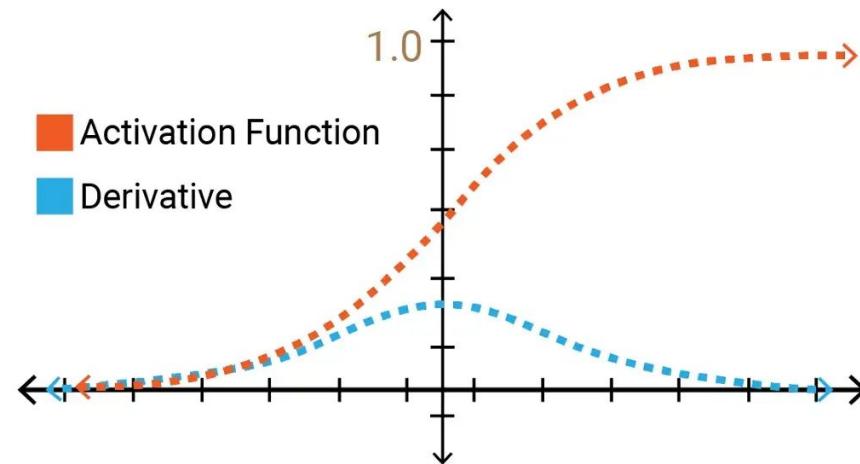
- Normalización de los datos de entrada
- Batch Normalization en las capas
- Intentar mitigar el problema del desvanecimiento del gradiente
- Intentar mitigar El problema de la explosión del gradiente
- Controlar la Inicialización de los pesos



Problemas de convergencia

El problema del desvanecimiento del gradiente y el problema de la explosión del gradiente se deben al error generado en cada capa durante la retro-propagación.

Función sigmoid y derivada

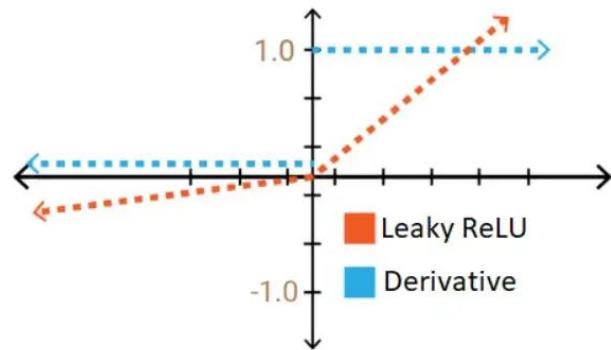


Cuando hay grandes cambios en el input de la función sigmoid, causará que los cambios en el output sean pequeños y por tanto las derivadas pequeñas. Cuando n hidden layers usan una función de activación como esta, se multiplican n derivadas que son pequeñas para actualizar los pesos. Esto lleva a que el gradiente decrece exponencialmente a medida que nos propagamos lejos de las capas finales (desvanecimiento del gradiente).

En el caso contrario de que los cambios sean grandes en el input, causará que el gradiente crezca exponencialmente (explosión del gradiente).

Problemas de convergencia

El problema del desvanecimiento del gradiente y el problema de la explosión del gradiente : cómo evitarlos.



Soluciones son:

- Usar la ReLU o similares en vez de la sigmoid
- Bach normalization
- Inicializar los pesos de la red no usando ceros o valores aleatorios.

Problemas de convergencia

Es necesario inicializar los pesos correctamente para que no se produzca el Vanishing Gradient o el Exploding Gradient problem.

- Cero
- Distribución aleatoria (normal)
- Xavier o Glorot: establece los pesos iniciales de cada capa de la red neuronal de forma aleatoria y uniforme, pero ajusta la escala de la distribución de inicialización para que sea proporcional a la raíz cuadrada de $1/n$, donde n es el número de neuronas en la capa anterior.

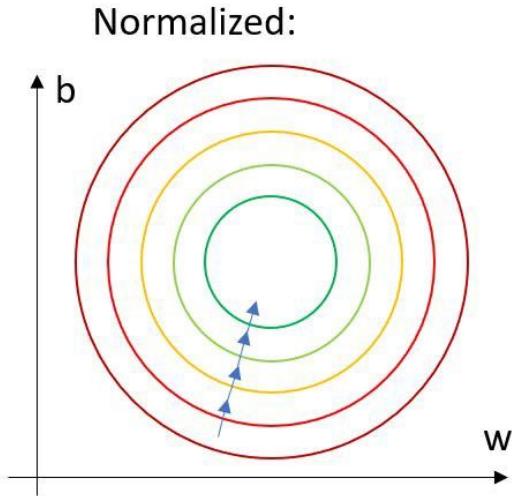
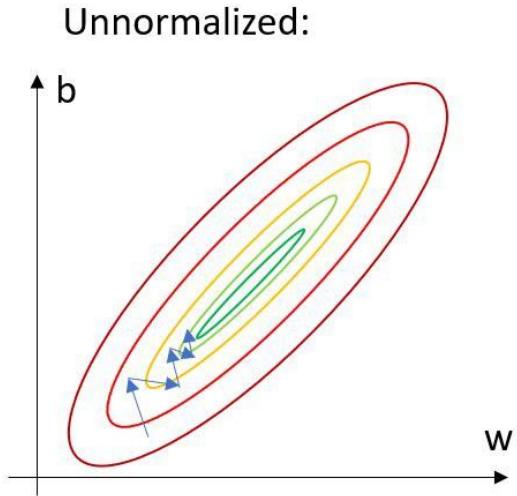
$$\text{Xavier o Glorot: } v = \sqrt{\frac{6}{fan_in + fan_out}}$$

fan_in = Número de inputs en la neurona

fan_out = Número de outputs de la neurona

Problemas de convergencia

Normalización de los datos de entrada.



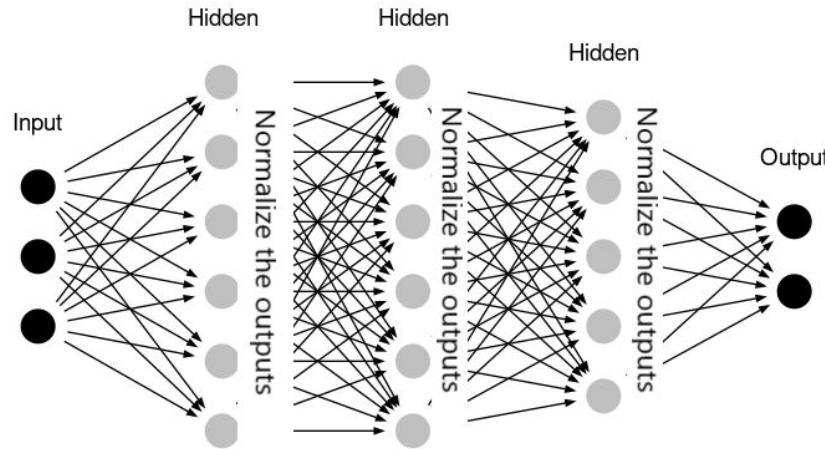
$$\mu = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2$$

$$\hat{x} = \frac{x - \mu}{\sigma^2}$$

Problemas de convergencia

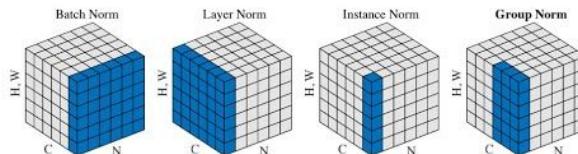
Normaliza la salida de la capa anterior restando la media del batch y dividiendo la desviación estándar.



Ventajas:

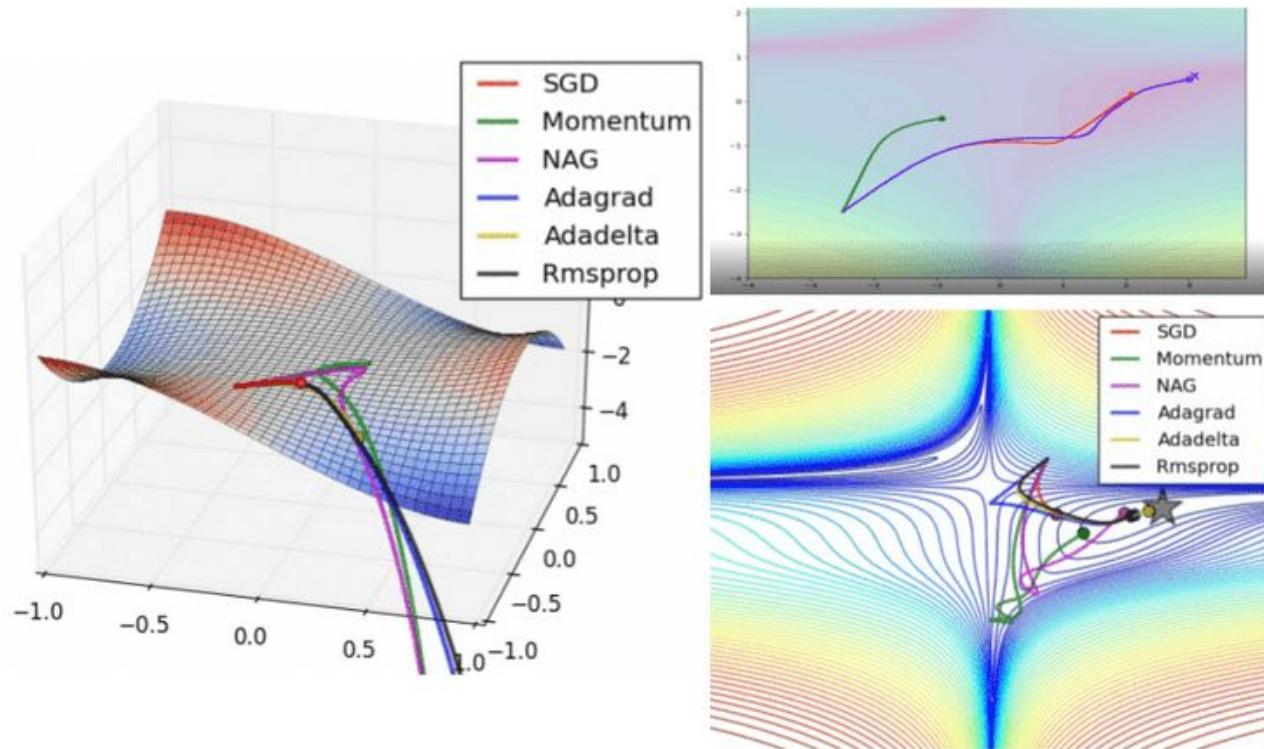
- Permite a cada capa de la red aprender por sí misma de una manera un poco más independiente.
- Podemos utilizar tasas de aprendizaje más altos ya que el batch normalization asegura que las activaciones no van tener valores muy altos o muy bajos.
- Reduce el sobre-ajuste.

Otros



Algoritmos de optimización

Métodos que permiten encontrar de la manera más óptima los pesos adecuados.



En el entrenamiento de las redes neuronales se busca minimizar la función de coste encontrando los pesos adecuados. El descubrimiento de estos pesos se lleva a cabo mediante el *backpropagation*. El optimizador, es el encargado de ir buscando estos pesos. Su funcionamiento esencial se basa en calcular el gradiente de la función de coste (derivada parcial) por cada peso (parametro/dimension) de la red.

Algoritmos de optimización

Inicialización de los pesos

Momentum

Acelera el DG en la dirección relevante y amortigua oscilaciones.

Nesterov

Tiene noción hacia dónde va, de manera que es capaz de frenar antes de que haya una colina y suba.

Adagrad

Adapta la tasa de aprendizaje a los parámetros: Actualizaciones más pequeñas para los asociados a características más frecuentes.

Adadelta

Extensión de Adagrad. Menos agresivo.

Adam

Método que adapta la tasa de aprendizaje a cada parámetro (como Adagrad)

Variantes Adagrad y Adam

Existen múltiples variantes de los algoritmos de optimización

Hiperparámetros

En los modelos de DL existen muchos hiperparámetros.

- Tasa de aprendizaje
- Número de capas ocultas
- Número de neuronas en cada capa
- Tamaño del mini-batch
- Algoritmo de optimización
- Etc...

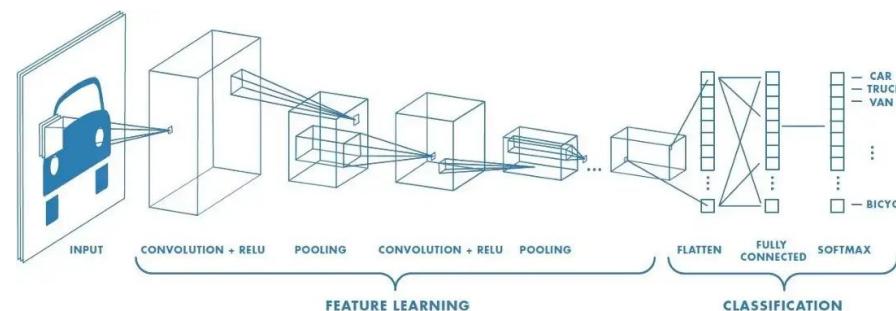
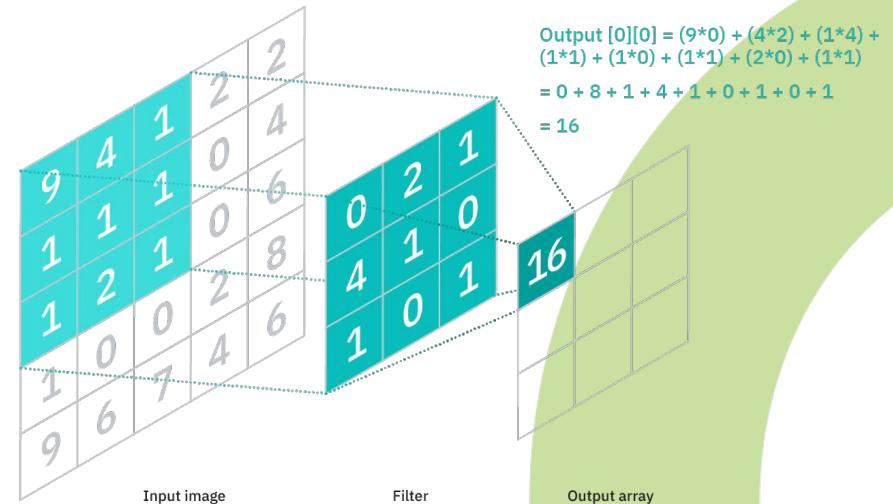
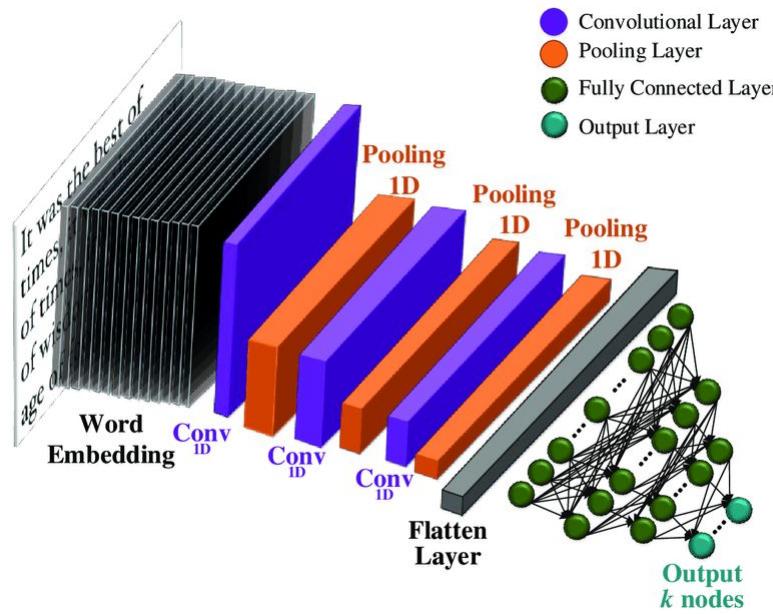
Índice de la sesión

- Recap
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- **Modelos de Deep Learning**
 - CNN
 - RNN
 - LSTM
 - GANS
 - Autoencoders



CNN

Redes convolucionales (CNN). Las redes convolucionales son utilizadas en modelos de clasificación de texto.



CNN

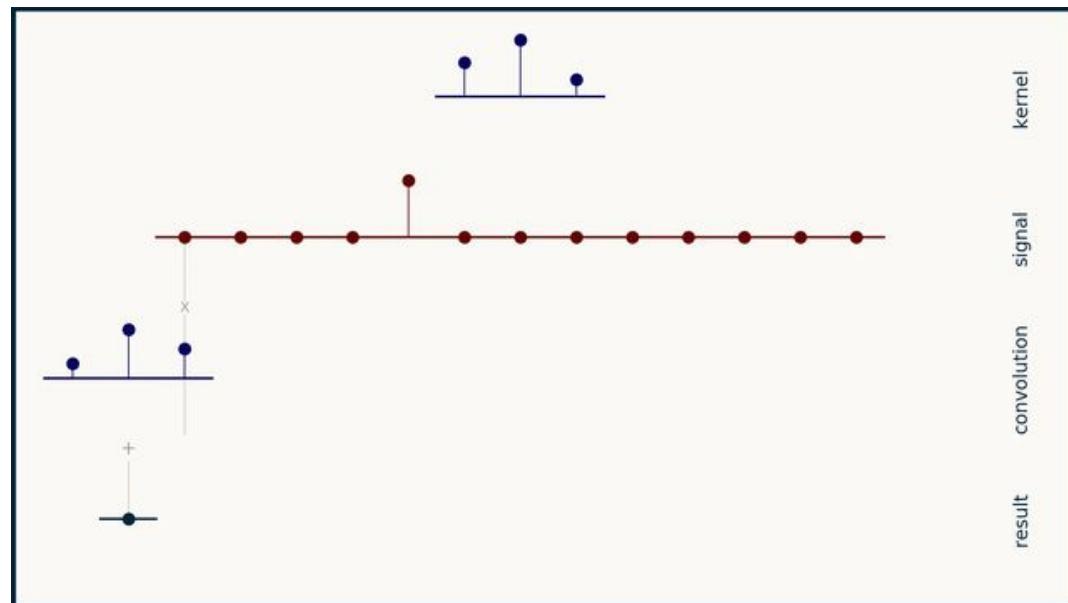
Ejemplo kernels.



<https://setosa.io/ev/image-kernels/>

CNN

Las CNN son capaces de captar información en cuadrantes, en vez de que cada input sea independiente. Hay convoluciones para varias dimensiones.



0	0	0	0	0	0	0
0	105	102	100	97	96	
0	103	99	103	101	102	
0	101	98	104	102	100	
0	99	101	106	104	99	
0	104	104	104	100	98	

Image Matrix

0	-1	0
-1	5	-1
0	-1	0

$$\begin{aligned} 0 * 0 + 0 * -1 + 0 * 0 \\ + 0 * -1 + 105 * 5 + 102 * -1 \\ + 0 * 0 + 103 * -1 + 99 * 0 = 320 \end{aligned}$$

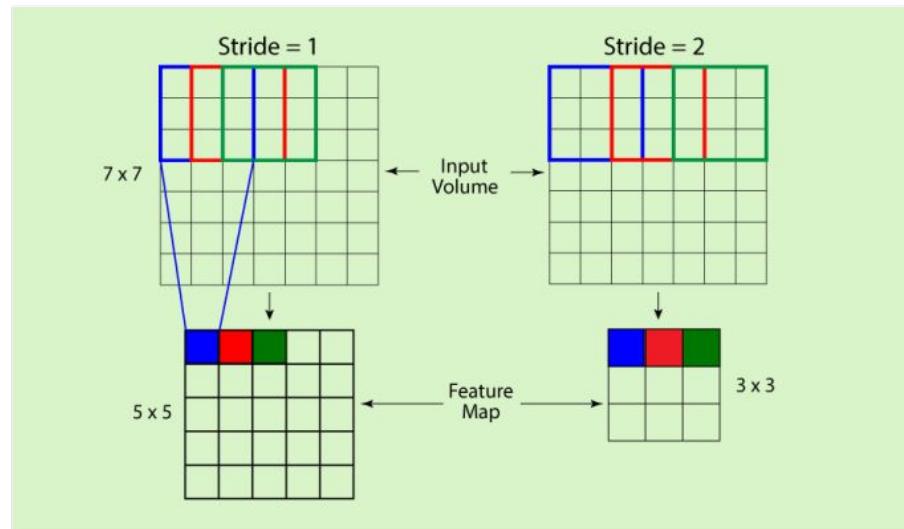
320			

Convolution with horizontal and vertical strides = 1

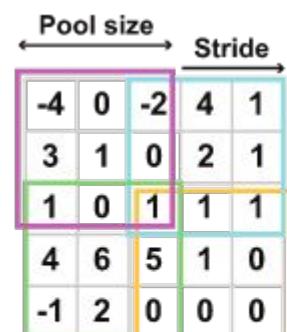
CNN

Usar diferentes filtros en cada capa aumenta la dimensionalidad, por lo que se usan métodos de pooling.

Strides



Pooling



Features

Max Pooling

3	4
6	5

Min Pooling

-4	-2
-1	0

Average Pooling

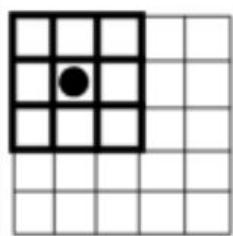
0	1
2	1

Output

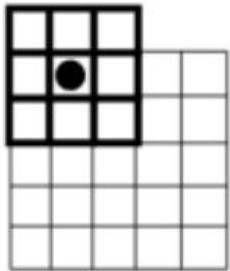
CNN

Conceptos más importantes de las CNN.

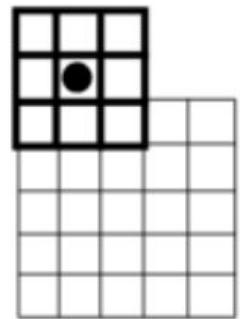
Capas CNN



VALID



SAME



FULL

Zero Padding

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	0	0	0
0	0	6	7	8	9	10	0	0	0
0	0	11	12	13	14	15	0	0	0
0	0	16	17	18	19	20	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Mirror Padding

13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8
3	2	1	2	3	4	5	4	3
8	7	6	7	8	9	10	9	8
13	12	11	12	13	14	15	14	13
18	17	16	17	18	19	20	19	18
13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8

Padding

Replicate Padding

1	1	1	2	3	4	5	5	5	5
1	1	1	2	3	4	5	5	5	5
1	1	1	2	3	4	5	5	5	5
6	6	6	7	8	9	10	10	10	10
11	11	11	12	13	14	15	15	15	15
16	16	16	17	18	19	20	20	20	20
16	16	16	17	18	19	20	20	20	20
16	16	16	17	18	19	20	20	20	20

CNN

Dimensiones y canales

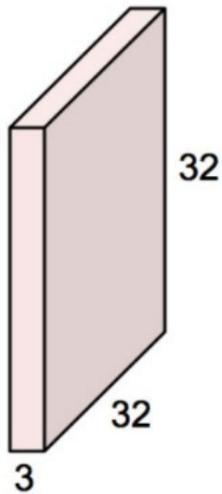


Imagen de entrada (lado x alto x canales)
 $32 \times 32 \times 3$



CNN

Dimensiones y canales

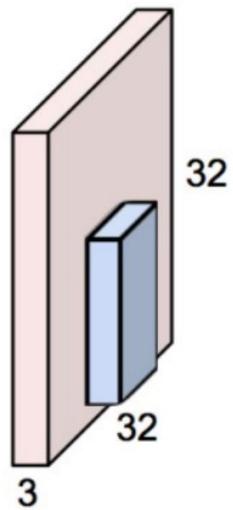


Imagen de entrada (lado x alto x canales)
 $32 \times 32 \times 3$

Filtro (kernel) 5x5 (lado x alto x canales)
 $5 \times 5 \times 3$

CNN

Dimensiones y canales

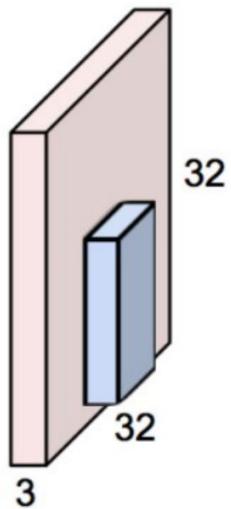


Imagen de entrada (lado x alto x canales)
32 x 32 x 3

Filtro (kernel) 5x5 (lado x alto x canales)
5 x 5 x 3

Número parámetros capa 1: Número de
filtros x dim filtro. P. e 6 filtros
6 x [5 x 5 x 3 + 1 (bias)]-> 456
parámetros

Convolución tipo Valid (P=0), Strides =1

Salida (lado x alto x canales) 28 x 28 x 6

Tamaño del Output

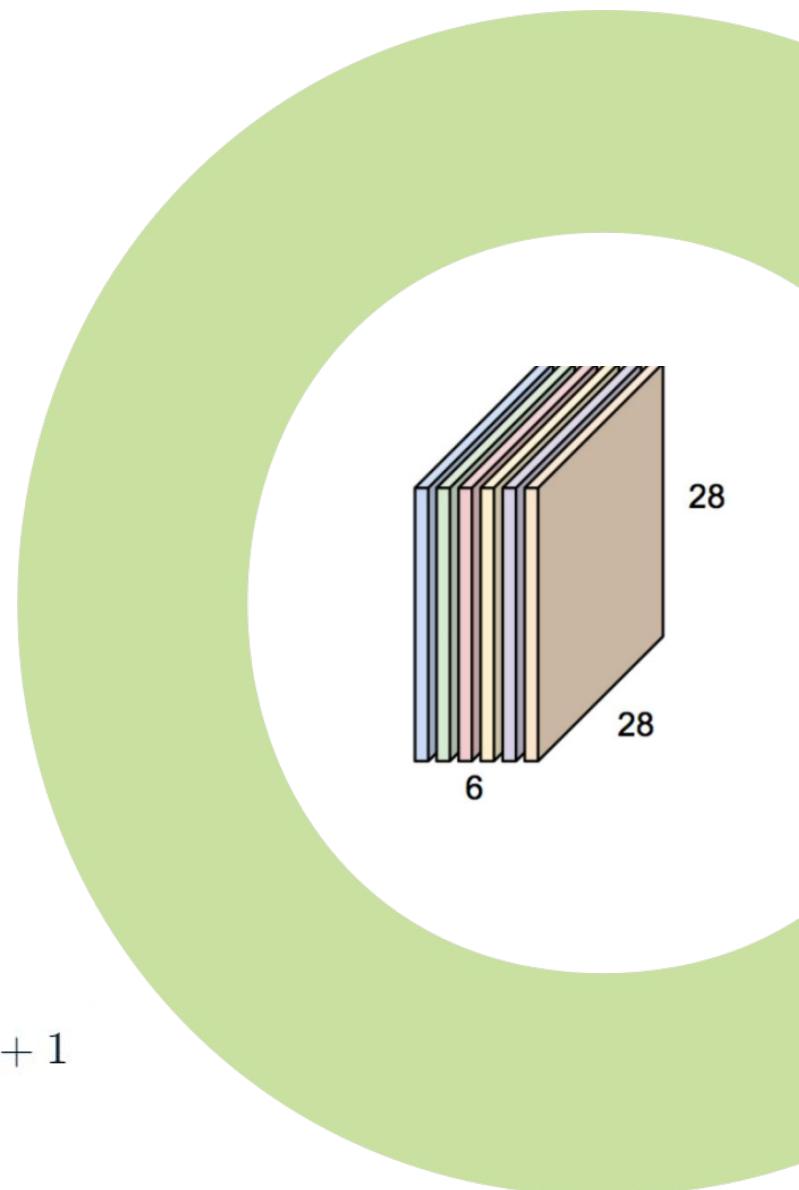
I = tamaño del input

K = tamaño del kernel

P = tamaño del zero padding

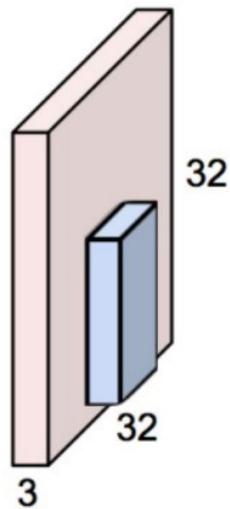
S = strides

$$O = ((I - K + 2P)/S) + 1$$



CNN

Ejercicio práctico



Si partimos de una Imagen de entrada de $64 \times 64 \times 3$ y le implementamos 3 filtros de 4×4 , ¿Cuál será el número de parámetros y el tamaño de salida de la capa?



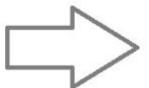
CNN

Al final se convierte el convolución en un vector.

Flatten

De convolución a Dense

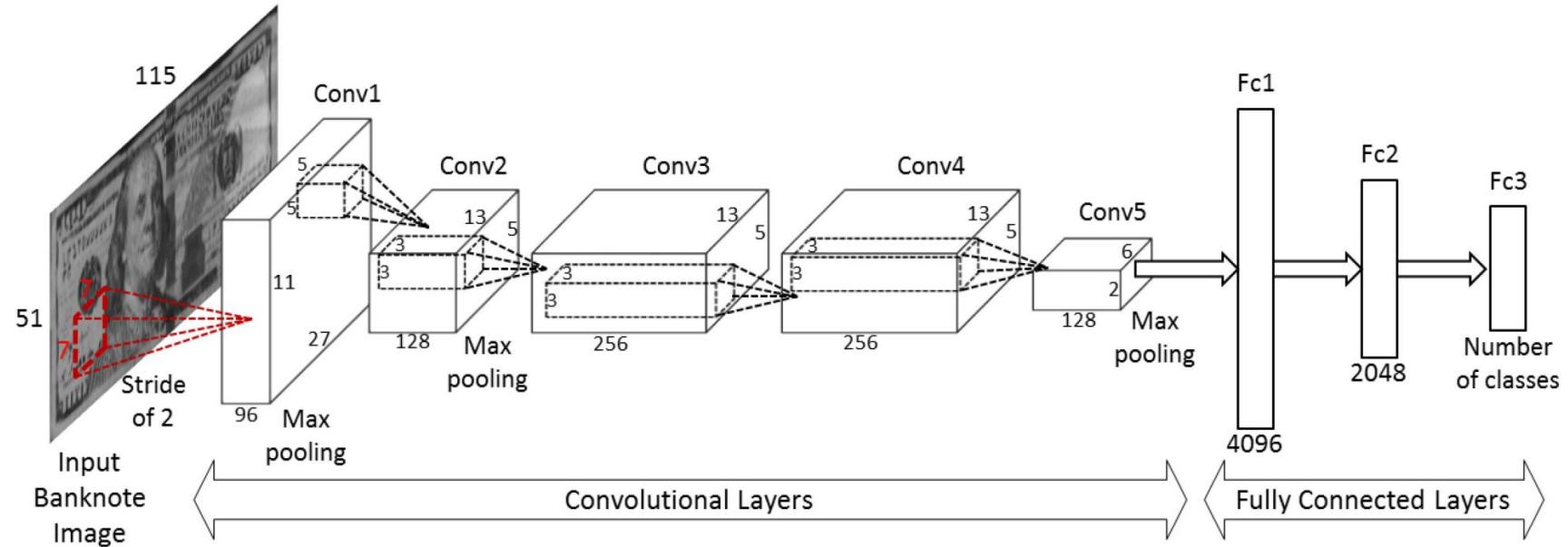
1	1	0
4	2	1
0	2	1



1
1
0
4
2
1
0
2
1

CNN

Conceptos más importantes de las CNN.



CNN

Ejemplo keras.

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten

model = Sequential()
model.add(Conv2D(10, kernel_size=3, activation='relu',
                input_shape=(img_rows, img_cols, 1)),padding="valid")

model.add(Flatten())
model.add(Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_data, train_labels, validation_split=0.2,epochs=3)

model.evaluate(test_data, test_labels)
```

CNN

Ejercicio práctico. Haz un modelo que prediga las 10 posibles clases que puede contener la imagen.

```
from keras.datasets import mnist
#download mnist data and split into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)

from keras.utils import to_categorical
#one-hot encode target column
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]
```

Crea a partir de los datos anteriores un modelo de CNN que tenga dos capas convolucionales:

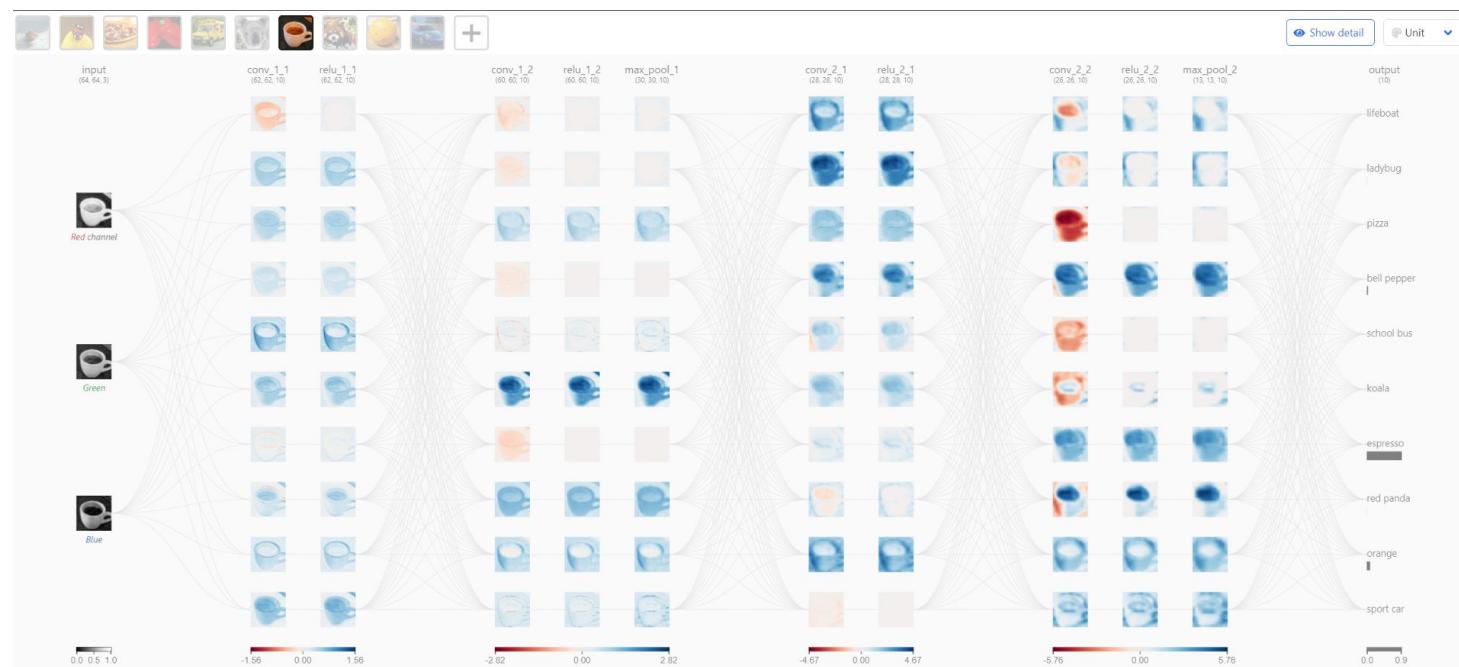
1. 64 filtros, kernel de 3 y activación 'relu'
2. 32 filtros, kernel de 3 y activación 'relu'

Recuerda hacer el Flatten a posteriori y añadir la capa densa con el output del modelo.

CNN

Ejemplo.

<https://poloclub.github.io/cnn-explainer/>



Índice de la sesión

- Recap
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- **Modelos de Deep Learning**
 - CNN
 - **RNN**
 - LSTM
 - GANS
 - Autoencoders



RNN

Redes recurrentes (RNN). Se usan cuando existe una dependencia secuencial.

Modelos para datos con dependencia temporal o secuencial ya que las RNN pueden capturar esta mediante la introducción de la "memoria".

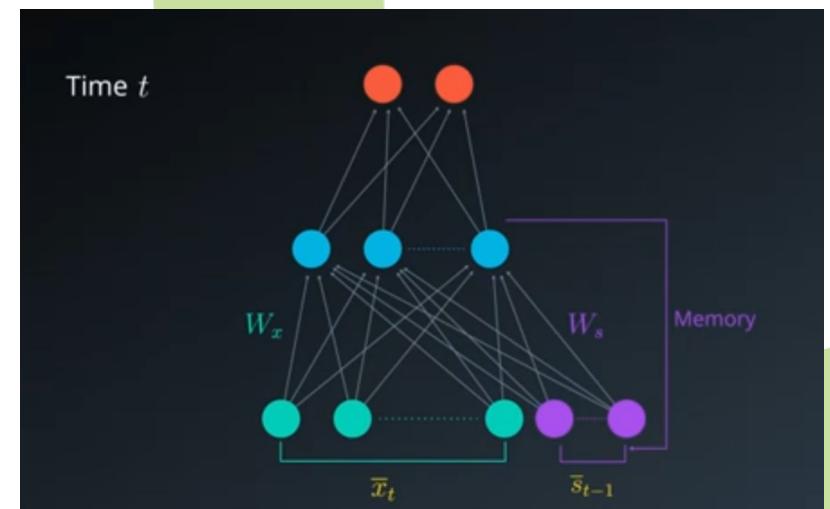
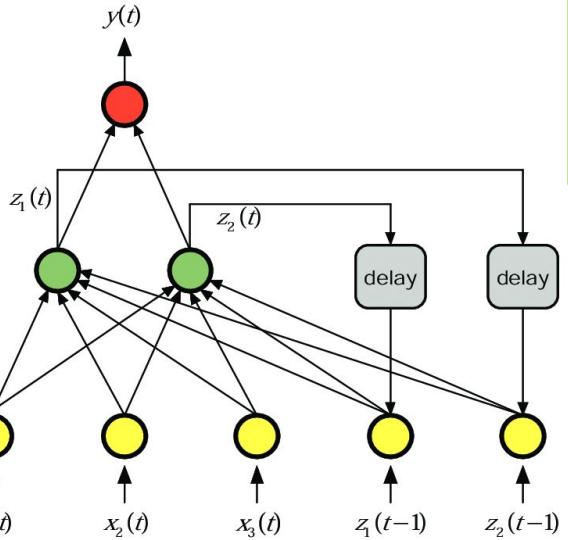
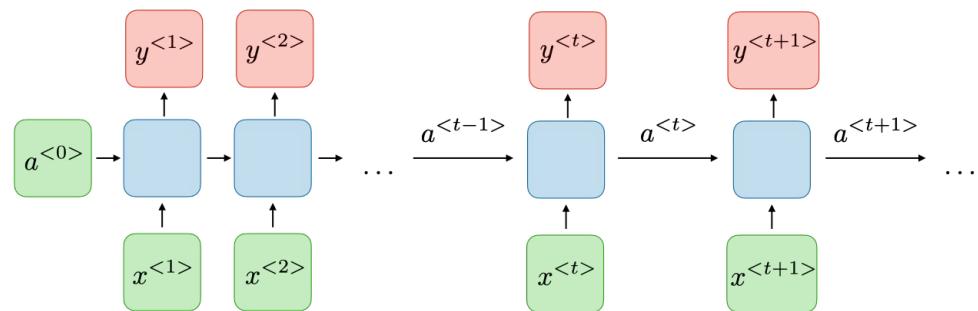
El concepto de recurrente viene debido a que se realiza la misma tarea sobre cada elemento del input de entrada.

Las RNN buscan mantener información de los inputs previos mediante los elementos de memoria denominados Estados. Los estados permiten recordar información corta (short term memory) debido al problema de "vanishing gradient".

La principal limitación que tiene es el denominado "vanishing gradient", de manera que el valor del gradiente se va haciendo más y más pequeño. Este problema es resuelto mediante las LSTM.

Usado para:

- Speech recognition
- Time series Prediction
- NLP (chatbots, question answering, ML Translation, predecir la siguiente palabra de una oración...)



Índice de la sesión

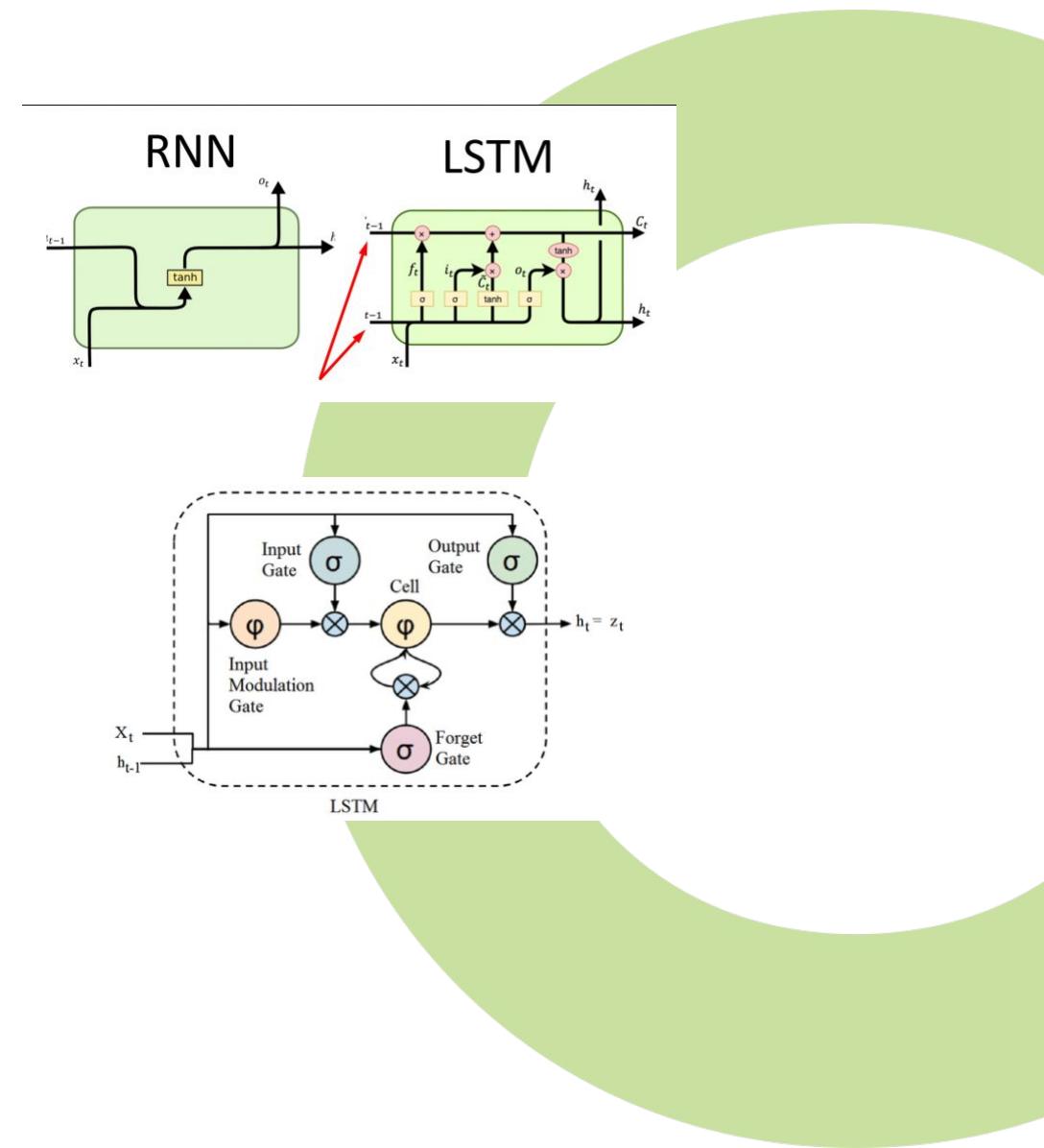
- Recap
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- **Modelos de Deep Learning**
 - CNN
 - RNN
 - **LSTM**
 - GANS
 - Autoencoders



LSTM

Long Short Term memory networks (LSTM).

- Las LSTM son una variación de las RNN que permiten aprender dependencias a largo plazo a diferencia de estas últimas.
- Las LSTM permiten evitar problemas de la RNN con el "vanish gradient descent" y captar mejor la dependencia secuencial a largo plazo. La principal diferencia es la función que define el perceptrón.
- No sólo considera el short-term memory, sino que añade términos que le permiten recordar a largo plazo.
- Las LSTM se componen de una célula de memoria y tres unidades reguladoras llamadas puertas ("gates"). Las LSTM añaden o eliminan información de las células en función del output de las puertas. La estructura de compuertas permite regular la información que se almacena. Generalmente hay 3 puertas: de entrada, salida y de olvido, cada una de ellas basada en una sigmoidal. La de forget se centra en qué información tiene que olvidarse mientras que la primera y segunda permiten controlar entrada y salida.



Índice de la sesión

- Recap
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- **Modelos de Deep Learning**
 - CNN
 - RNN
 - LSTM
 - **GANS**
 - Autoencoders



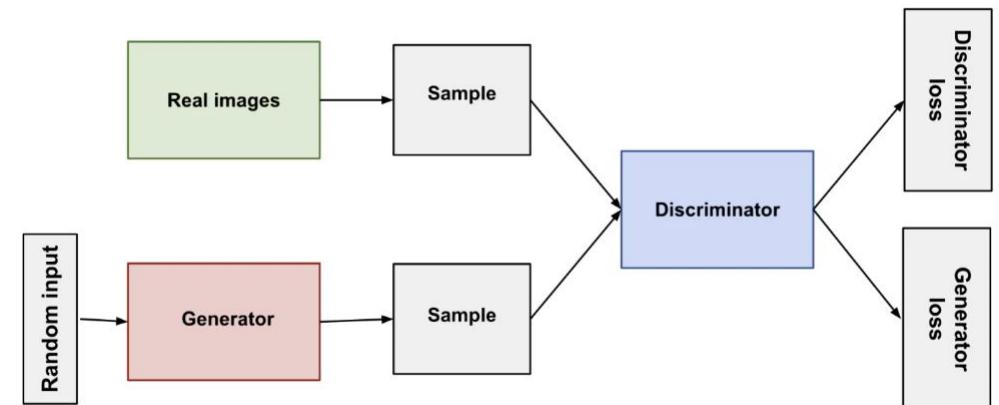
Gans

Las GANs son modelos compuestos por un generador y un discriminador que compiten entre sí en un juego de suma 0 para realizar una tarea.

La dinámica del juego en las GAN se realiza mediante dos redes neuronales: el generador y el discriminador:

- El generador aprende a generar datos plausibles. Las instancias generadas se convierten en ejemplos de entrenamiento negativos para el discriminador.
- El discriminador aprende a distinguir los datos falsos del generador de los datos reales. El discriminador penaliza al generador por producir resultados inverosímiles.

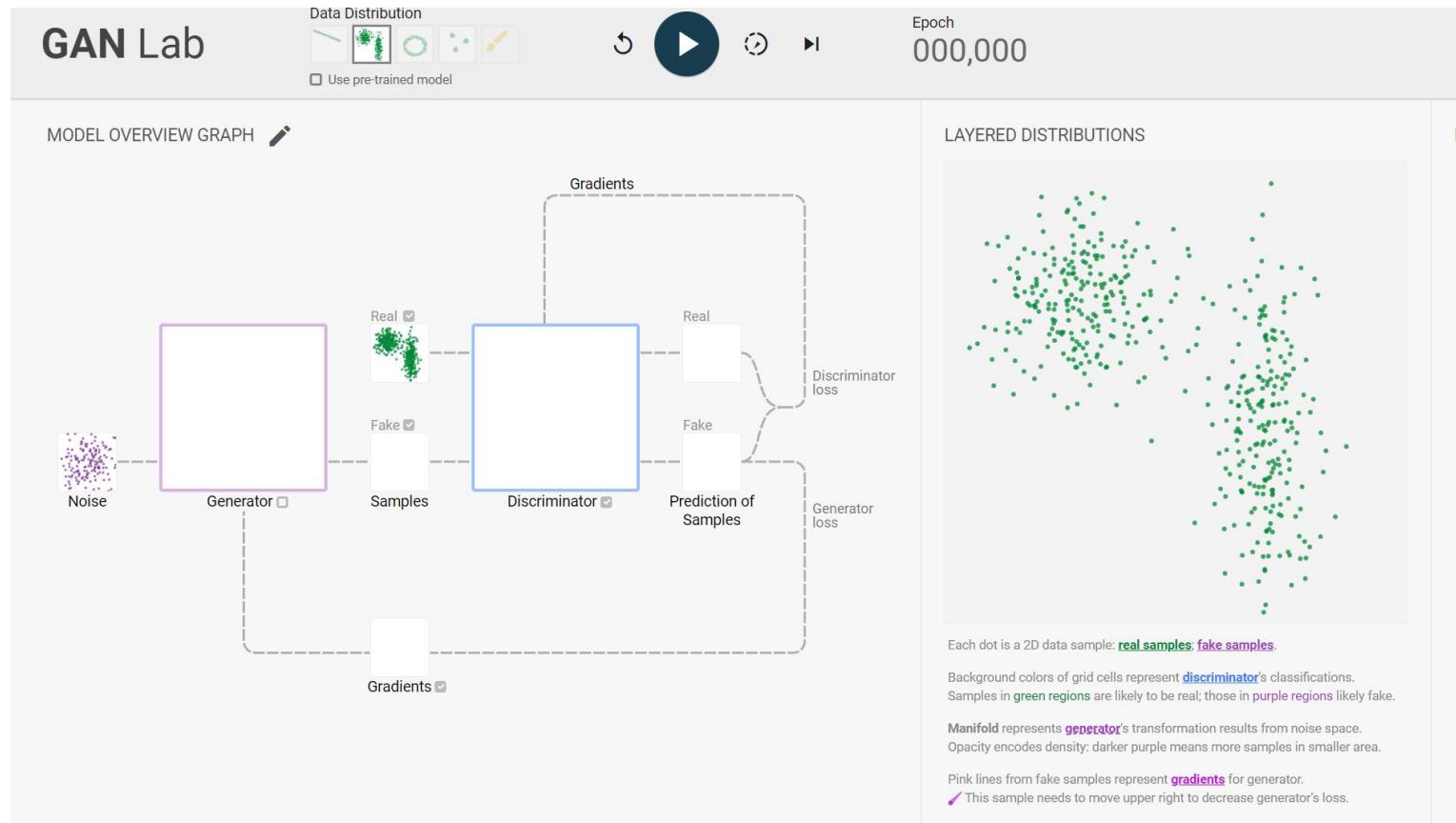
Las GANs fueron introducidas en 2014 en un paper de Ian Goodfellow. Uno de los desafíos discutidos en el documento estaba relacionado con el training. Debido a que las GAN involucran dos redes neuronales, también requieren dos procesos de entrenamiento diferentes. De manera similar, las GAN necesitan evaluar dos funciones de pérdida en lugar de una para replicar la distribución de probabilidad del conjunto de datos original. Esta sigue siendo un área activa de investigación dentro del espacio GAN.



Gans

GansLab

<https://poloclub.github.io/ganlab/>



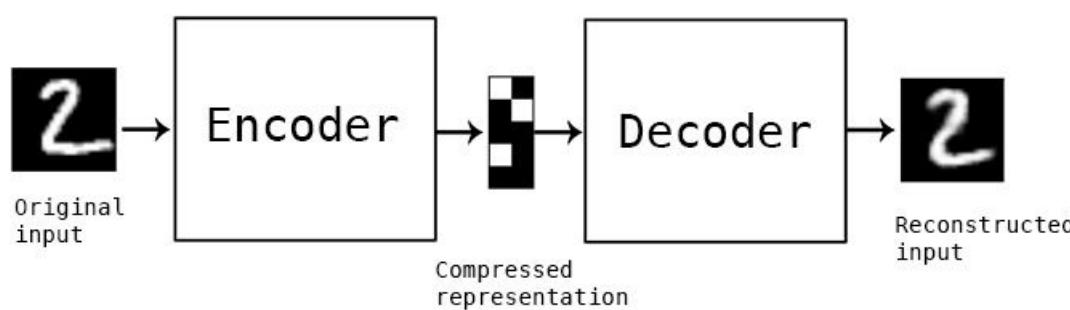
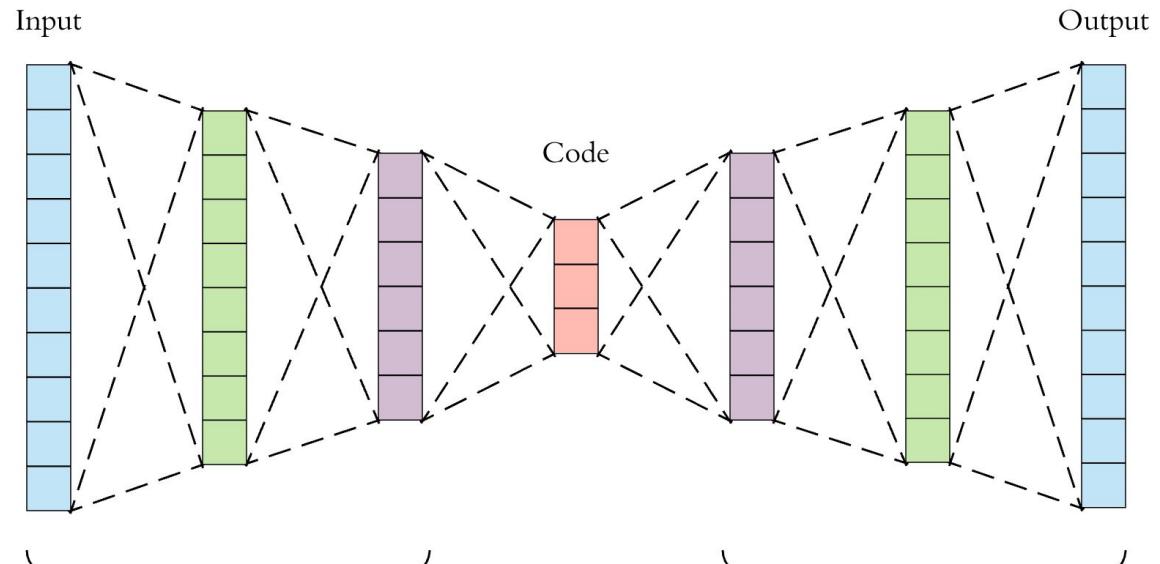
Índice de la sesión

- Recap
- Introducción a las redes neuronales
- Configuración de los modelos de redes neuronales
- **Modelos de Deep Learning**
 - CNN
 - RNN
 - LSTM
 - GANS
 - **Autoencoders**



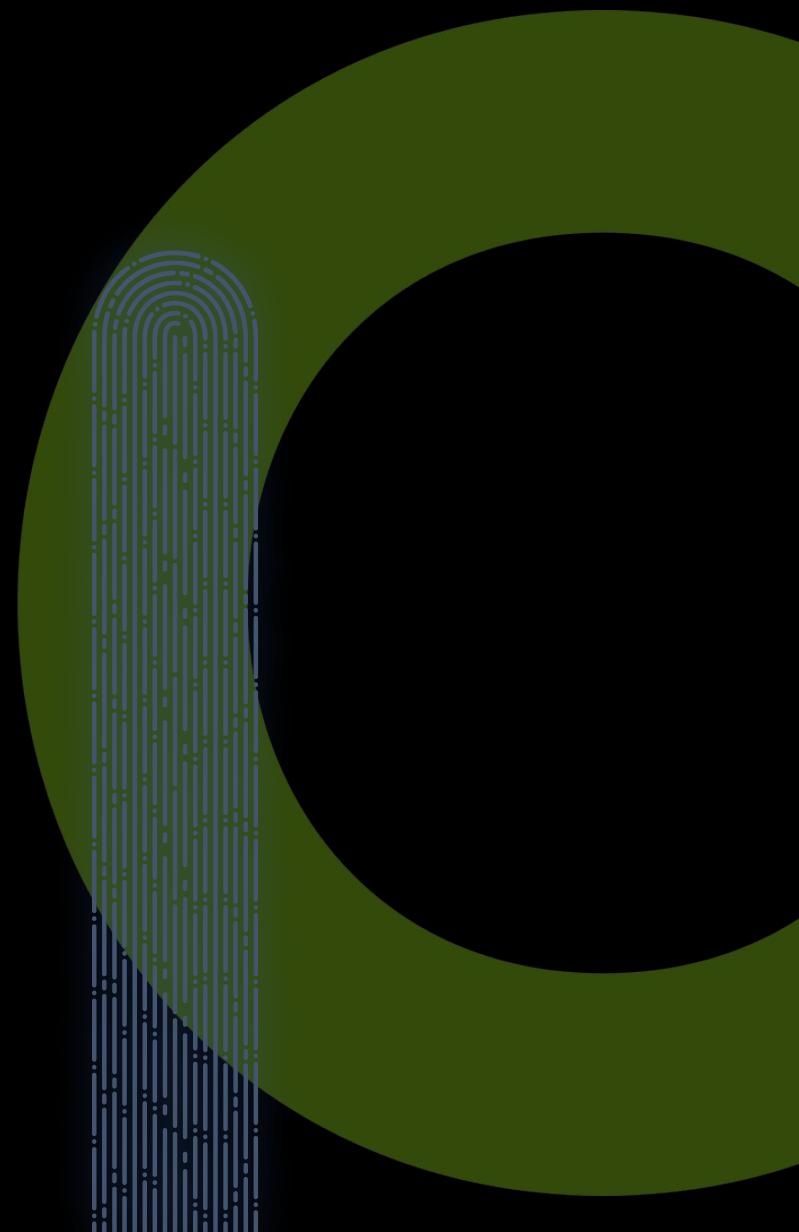
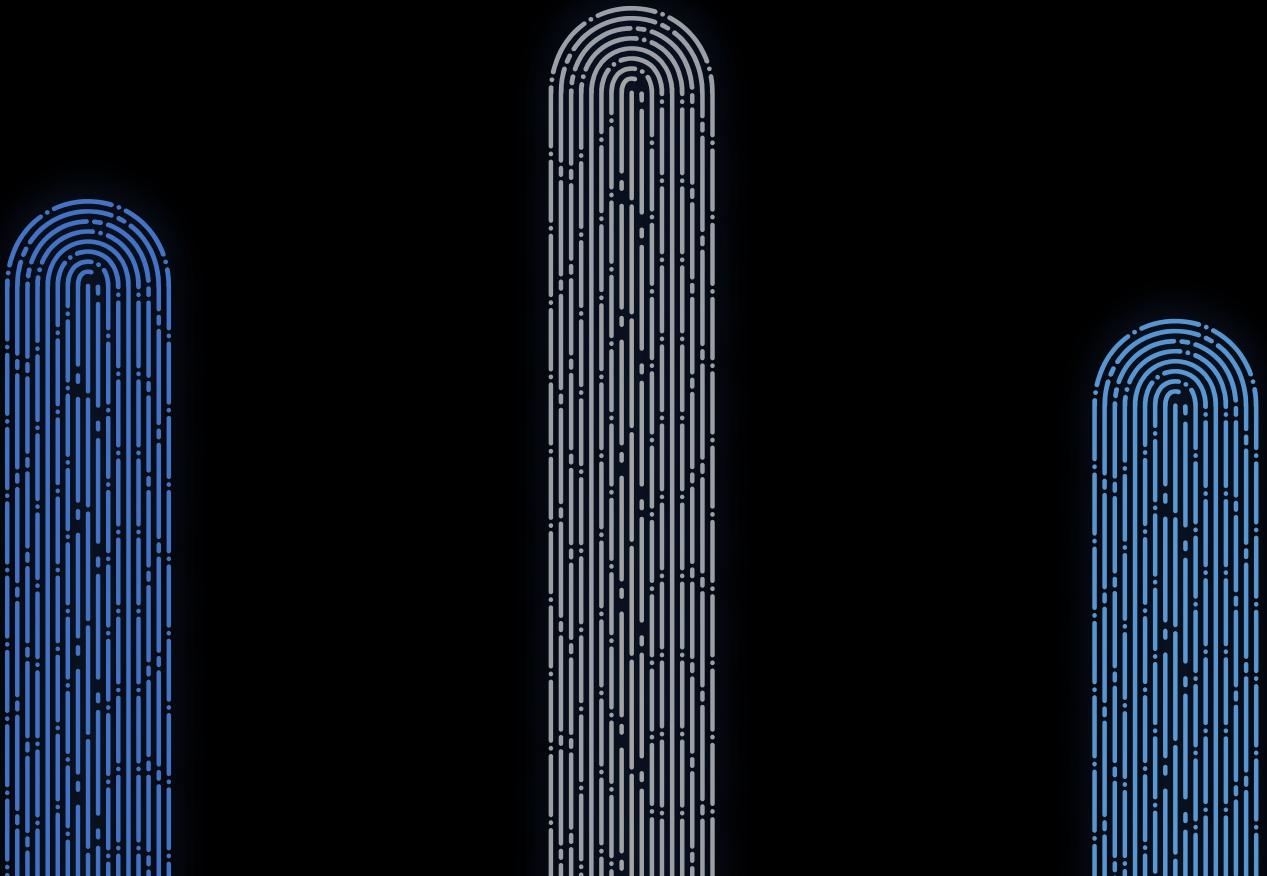
AutoEncoders

Los autoencoder son entrenados de manera no supervisada y están compuestos por un encoder y un decoder.



Los autoencoders tienen como objetivo aprender primero las representaciones codificadas de nuestros datos y luego generar los datos de entrada (lo más cerca posible) a partir de las representaciones codificadas aprendidas. Por tanto, la salida de un autoencoder es su predicción para la entrada.

Muchas gracias!



Ejercicio práctico

Titanic: Modelo que prediga si un pasajero sobrevive en función de las siguientes variables.

PassengerID	ID del pasajero
Survived	Si sobrevivió
Name	Nombre
Pclass	Clase del ticket
Sex	Sexo
Age	Edad
SibSp	Nº de hermanos/esposos a bordo
Parch	Nº de padres/hijos a bordo
Ticket	Nº de ticket
Fare	Tarifa
Cabin	Nº de cabina
Embarked	Puerto de embarque

